MASTER EN SCIENCES INFORMATIQUES

Inégalité de Hoeffding pour la classification en machine learning

Joskin, Pierre

Award date:
2018

Awarding institution:
Universite de Namur

Université de Namur
Faculté d'informatique
Année académique 2017–2018

Inégalité de Hoeffding pour la classification en machine learning

Pierre Joskin

Maître de stage : Professeur Benoît Frénay

Promoteur : _____ (Signature pour approbation du dépôt - REE art. 40)
Professeur Benoît Frénay

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

Table des matières

Remerciements

J'aimerais tout d'abord remercier mon promoteur Mr. Benoît Frénay, pour ses conseils avisés, les brainstormings stimulants et la patience dont il a fait preuve.

J'aimerais remercier ma famille et mes amis pour leurs encouragements. Et particulièrement Yves Joskin, mon père, pour ses relectures et ses remarques pertinentes.

J'aimerais remercier Lauriane Castin pour sa relecture attentive et ses remarques constructives.

J'aimerais également remercier Mme Benjamine Lurquin et tout le secrétariat de la Faculté d'informatique de l'Université de Namur sans qui je me serais noyé dans l'administratif nécessaire aux études universitaire.

J'aimerais également remercier tous ceux qui donnent libre accès à leurs données et leurs expériences, en particulier l'hôpital universitaire de Wisconsin-Madison pour deux de leurs datasets que j'utilise dans ce mémoire.

Chapitre 1

Introduction

Ce mémoire est réalisé dans le contexte d'un master en informatique en horaire décalé à l'Université de Namur. Il traite de sujets liés au machine learning et au big data. Dans ce domaine, un grand nombre de questions sont toujours ouvertes et en cours de recherche.

L'utilisation des arbres de décision sur des données provenant du streaming est abordée. Dans ce cadre, la création d'un arbre demande une quantité de mémoire tellement importante qu'il s'avère parfois qu'un ordinateur ne peut pas la gérer. De plus il peut arriver que l'arbre ainsi créé soit lent à l'utilisation, voire qu'il ne puisse pas jouer son rôle de classificateur correctement. Une solution pour répondre à ces problèmes est exposée dans ce mémoire. L'algorithme des plus proches voisins est également abordé sur la problématique de la détermination du nombre de voisins à utiliser lors de la classification. La question de la quantité de données nécessaire à cet algorithme pour fonctionner est également étudiée.

Pour adresser ces problèmes, la relation nommée « inégalité (ou borne) de Hoeffding [10] » est étudiée comme formule mathématique sur laquelle se basent les solutions proposées. Elle permet de déterminer une probabilité liée à la somme de variables aléatoires indépendantes par rapport à la valeur véritable que les variables devraient avoir. Dans le cadre présent, les variables sont des probabilités et la formule est utilisée dans l'algorithme permettant de déterminer si une décision sur ces probabilités peut être prise ou pas. Les probabilités sont calculées sur des données devant être classifiées avec des techniques de machine learning. L'inégalité de Hoeffding se base sur une somme, ce qui permet de l'utiliser dans un cadre incrémental en ajoutant des données à la somme petit à petit. Grâce à cela, son utilisation se prête bien

aux systèmes travaillant sur un grand nombre de données comme le big data.

Le chapitre *Les concepts* présente les différents concepts qui seront utilisés dans la suite de ce mémoire, à savoir le machine learning, le big data et l'inégalité de Hoeffding. Le chapitre *Les algorithmes « Very Fast »* répond au problème lié aux arbres de décision. Il discute de l'algorithme proposé par Pedro Domingos et Goeff Hulten, le very fast decision tree[6]. Plus généralement, il expose les algorithmes dits « very fast » qui en utilisent les principes. Ensuite, le mémoire présente deux contributions originales qui utilisent les principes déjà employés dans le very fast decision tree, pour tenter de résoudre les problèmes soulevés dans l'algorithme des plus proches voisins. La première contribution, décrite dans le chapitre *Le k local nearest neighbors (kLNN),* propose de définir un ensemble de valeurs pour le nombre de voisins à prendre en compte lors de la classification d'une donnée dépendante de la zone où se trouve la donnée à classifier. L'objectif est de fournir un procédé automatique justifié mathématiquement pour définir le nombre d'éléments à prendre en compte dans l'algorithme des plus proches voisins. La seconde proposition, exposée au chapitre *Le kNN réduit,* propose une technique visant à réduire la quantité de données à retenir pour créer un modèle viable. L'avantage d'un modèle plus petit est qu'il est plus rapide à exécuter et moins gourmand en mémoire. C'est particulièrement intéressant dans le cas de modèles très grands au départ.

Chapitre 2

Les concepts

Ce chapitre présente trois concepts clés utilisés dans ce mémoire. Le premier est le machine learning, tous les algorithmes discutés dans ce document sont abordés et appliqués dans ce domaine particulier de l'informatique. Le second est le big data, un des objectifs des algorithmes discutés dans ce document est de pouvoir être utilisé ou de pouvoir améliorer leur utilisation dans ce domaine. Et enfin, l'inégalité de Hoeffding[10], il s'agit d'une formule mathématique à la base de tous les algorithmes discutés dans ce document. D'autres concepts plus spécifiques à une sous-partie du mémoire seront présentés plus tard, tels que les arbres de décision et l'algorithme des plus proches voisins.

2.1 Le machine learning

Le machine learning est un ensemble de techniques informatiques qui permet de résoudre des questions de manière autonome en se basant sur l'apprentissage et l'amélioration continue sans devoir implémenter explicitement les logiques nécessaires à la résolution des questions. C'est une branche de l'intelligence artificielle.

Habituellement, le machine learning se déroule en deux phases. La première est la création d'un modèle de données grâce à l'observation d'un ensemble de données appelé dataset. La seconde est l'application du modèle sur de nouvelles données pour tenter de prendre une décision sur celles-ci. Dans ce mémoire, la décision recherchée est une classification, c'est-à-dire que le modèle exprime un ensemble de labels représentant les données. L'ap-

plication du modèle sur une nouvelle donnée donne le label auquel la donnée appartient. Par exemple, un modèle créé sur des images de légumes pourrait exposer les labels {*haricot, pomme de terre, salade*} et lorsque le modèle est appliqué à une image d'une pomme de terre, le résultat est *pomme de terre*.

Un grand nombre de techniques et d'algorithmes utilisent ces deux phases. Ils sont regroupés en fonction de la manière dont la première phase se déroule. De manière non-exhaustive, voici quelques-uns de ces groupes :

L'apprentissage supervisé

Lors de la création du modèle, les données du dataset sont étiquetées (labellisées) au préalable par un intervenant extérieur. L'application du modèle devra alors prédire le label correspondant à la donnée observée. Il y a donc une connaissance a priori des labels sur les données utilisées lors de la création du modèle. Les algorithmes discutés dans ce mémoire, les arbres de décision et les plus proches voisins, font partie de ce groupe. Par exemple, pour des images de légumes, chaque image utilisée pour créer le modèle doit, a priori, posséder une description contenant le nom du légume qu'elle représente. L'application du modèle à une nouvelle image donnera alors le nom du légume que le modèle a reconnu.

L'apprentissage non-supervisé

Lors de la création du modèle, les données du dataset ne sont pas étiquetées au préalable. La création du modèle devra alors définir automatiquement le nombre de labels à exposer ainsi que la manière de classifier les données dans ces labels ; par exemple, pour les images de légumes, l'algorithme devra détecter les différences et regrouper les images qu'il considère comme représentant le même légume. Ensuite, l'application du modèle donnera alors comme résultat le groupe auquel l'image correspond le mieux.

L'apprentissage semi-supervisé

Lors de la création du modèle, une majorité des données du dataset n'est pas étiqueté, et une minorité l'est. L'application du modèle est similaire à l'apprentissage supervisé

2.2 Le big data

Le concept de big data représente la manière d'aborder l'analyse de données répondant à certains critères. Ces critères sont résumés via les « 5 Vs » :

1. **Volume** : le big data concerne de grandes quantités de données, tellement grandes qu'il n'est parfois pas possible de les stocker tel quel.
2. **Variety** : le big data concerne une grande variété de données. Celles-ci peuvent être des données structurées, c'est-à-dire des données dont l'ordre a un sens, comme du texte brut ou des images. Elles peuvent également être des données non-structurées, c'est-à-dire des données dont l'ordre n'est pas important pour pouvoir l'interpréter, comme des tableaux ou des bases de données.
3. **Velocity** : le big data concerne des données qui sont générées rapidement, et qui doivent être traitées tout aussi rapidement. Ces données sont parfois éphémères, ne possédant qu'une durée de vie limitée.
4. **Veracity** : le big data concerne des données potentiellement fausses. Ces données ne peuvent pas être interprétées comme toujours vraies. Certaines données peuvent comporter des erreurs, incohérences et imprécisions.
5. **Value** : le big data tente de résoudre des tâches dépendant des données. Vu leur grande quantité, leur variété et leur vitesse, la transformation de celles-ci en données exploitables est primordiale.

Le domaine d'application du big data est quasi infini. Partout où il y a des données répondant aux critères des 5Vs qui transitent, le big data s'applique. Le trafic sur un site web, les transactions boursières, les données provenant des capteurs des chaines de montage ou les données échangées sur un réseau social sont tous des exemples de cas d'application.

Une manière d'utiliser les données provenant du big data est le machine learning. Les techniques de machine learning existant depuis bien avant le concept de big data, elles ne sont donc pas nécessairement adaptées aux défis intrinsèques du big data. Dans le chapitre 3, la discussion portera sur une manière permettant à l'algorithme des arbres de décision d'appréhender les données du big data.

2.3 L'inégalité de Hoeffding

Dans le domaine du machine learning, il arrive souvent que les problèmes puissent se résoudre grâce à des sommes comparées les unes aux autres. Par exemple, imaginons que, pour déterminer la préférence musicale d'une personne, il suffise de compter le nombre de chansons écoutées par cette personne pour chaque style musical et prendre le style le plus représenté. Les sommes dans ce cas-ci sont les comptes pour chaque style de musique.

Pour être sûr et certain de résoudre le problème posé correctement, il faut observer tous les échantillons disponibles. Dans l'exemple de la préférence musicale, il faudrait prendre en compte toutes les chansons que la personne a écoutées depuis toujours. La question liée au changement de goûts musicaux dans le temps est ignorée dans cet exemple ; ce problème est à adresser avec le phénomène de *drift* qui sera abordé au section §3.5. Néanmoins, est-il toujours nécessaire d'observer toutes les données pour arriver à prendre une décision ? La réponse est non. D'ailleurs, prendre en compte toutes les données lorsque l'on parle de big data, n'est pas possible. Il serait intéressant d'avoir un critère permettant de déterminer à partir de quel moment suffisamment d'éléments (de chansons dans l'exemple) ont été observés pour pouvoir déterminer avec certitude ce qui doit l'être (le style musical préféré dans l'exemple). De plus, si ce critère pouvait être agnostique de la sémantique de la donnée observée, cela permettrait de l'utiliser quel que soit le domaine d'application.

Le critère d'arrêt doit donc est spécifique aux données observées et à leur domaine, et il n'est pas possible d'en créer un qui soit totalement générique. Par contre, si la contrainte sur la certitude est relâchée, les mathématiques probabilistes fournissent des formules intéressantes. Parmi celles-ci, la formule proposée par Wassily Hoeffding [10] correspond bien au problème à résoudre.

L'inégalité de Hoeffding[10] fonctionne sur un principe de comparaison entre deux sommes, $S1$ et $S2$ représentant chacune un choix possible. Ces sommes sont notamment dépendantes du nombre de données observées. Si la comparaison est supérieure ou égale à une borne ϵ, alors le critère est atteint et l'algorithme peut s'arrêter : $|S_1 - S_2| \geq \epsilon$ avec une certaine certitude $(1 - \delta)$. Lorsque le critère est atteint, le choix à faire est celui lié à la somme la plus grande.

La borne de Hoeffding se calcule grâce à la formule

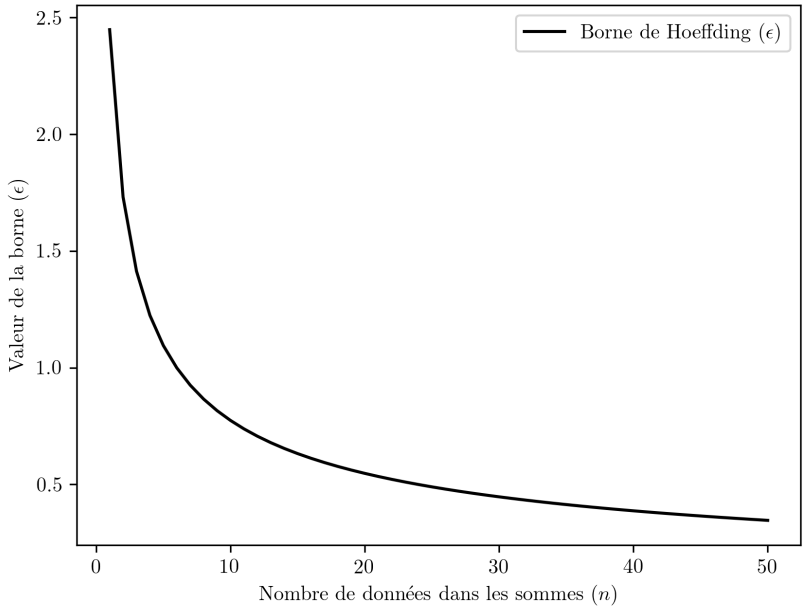


FIGURE 2.1 – Valeurs de la borne de Hoeffding pour $R = 1$ et $1 - \delta = 95\%$

$$\epsilon = 2\sqrt{\frac{R^2 ln(\frac{1}{\delta})}{2n}} \tag{2.1}$$

où R est la largeur du domaine d'application des éléments des sommes S_1 et S_2, δ la certitude que le critère est effectivement atteint et n le nombre d'éléments présents dans S_1 et S_2.

La figure 2.1 montre l'évolution de la borne de Hoeffding utilisée dans l'inégalité du même nom en fonction du nombre d'éléments présents dans les sommes à comparer. L'exemple montre ces valeurs pour $1 - \delta = 95\%$.

Dans l'exemple des styles de musiques, une somme par style de musique est calculé $(S_{rap}, S_{classique}, ...)$, et une fonction d'évaluation par somme est également utilisée $(Eval_{rap}, Eval_{Classique}, ...)$. Ces fonctions renvoient 1 sur la chanson est du style correspondant et 0 sinon. Concrètement, si les chansons écoutées par la personne sont observées dans l'ordre suivant

rap, classique, classique, rap, rap, classique, rap, rap, classique, classique, classique, rap, rap, rap, classique, rap, classique, rap, rap, rap, rap, rap, classique, rap, rap, rap, rap, rap, classique, rap, rap, rap, classique, rap, rap, rap, classique, rap, rap, rap, rap, rap, classique, rap, classique, rap, rap, rap, rap, rap

Alors l'évolution des probabilités que le style musical préféré soit rap ou classique au fur et à mesure des écoutes peuvent se représenter comme sur la figure 2.2. Il est alors possible de calculer la différence entre S_{rap}et $S_{classique}$. La figure 2.3 montre S_{rap}, $S_{classique}$, ϵ et $|S_{rap} - S_{classique}|$. L'intersection entre ϵ et $|S_{rap} - S_{classique}|$ est le moment où le critère est atteint et où il n'y a plus besoin de compter davantage de chansons pour connaitre avec certitude le style musical préféré. À partir de ces 40 observations, la borne de Hoeffding permet de dire que le rap est le style de musique préféré de la personne avec une certitude de 95%.

Il est intéressant de remarquer que si les premières des musiques écoutées étaient toutes du rap comme montré dans la figure 2.4, la détermination du style serait alors beaucoup plus rapide. De manière générale, lorsque les premières observations sont toutes les mêmes et donc qu'une des deux sommes est maximale et l'autre minimale, il est possible de déterminer un nombre minimal d'observations nécessaires directement grâce à la formule. En effet, dans ce cas là, $S_1 - S_2 = R$ et donc :

$$\epsilon = R = 2\sqrt{\frac{R^2 ln(\frac{1}{\delta})}{2n}} \tag{2.2}$$

$$R^2 = 4\frac{R^2 ln(\frac{1}{\delta})}{2n} \tag{2.3}$$

$$n = 2ln(\frac{1}{\delta}) \tag{2.4}$$

Dans l'exemple, dans le cas où toutes les premières chansons écoutées seraient du rap, avec $1 - \delta = 95\%$, donne $n = \lceil 5.99 \rceil = 6$ pour pouvoir déterminer que c'est le rap le style de musique préférée.

Cela a comme implication directe qu'il n'y a aucun intérêt à observer moins d'éléments que cette valeur minimale. Et donc, dans tous les algorithmes présentés dans ce mémoire, les sommes ne sont comparées entre elles qu'à partir de ce nombre minimal d'observations.

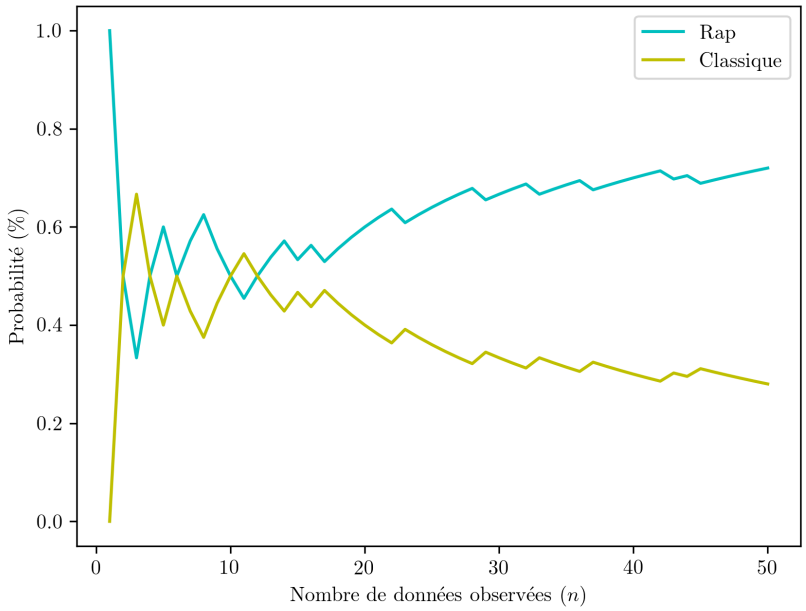


FIGURE 2.2 – Probabilités de préférence des styles musicaux.

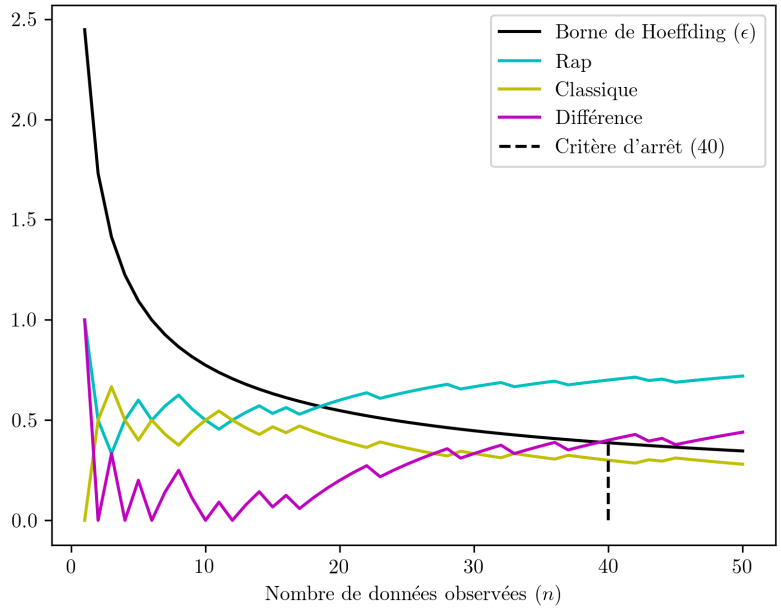


FIGURE 2.3 – Exemple d'application de la borne de Hoeffding. Avec une probabilité de $1 - \delta = 95\%$ et $R = 1$, la borne permet de déterminer le critère d'arrêt à la $40^{\text{ème}}$ observation.

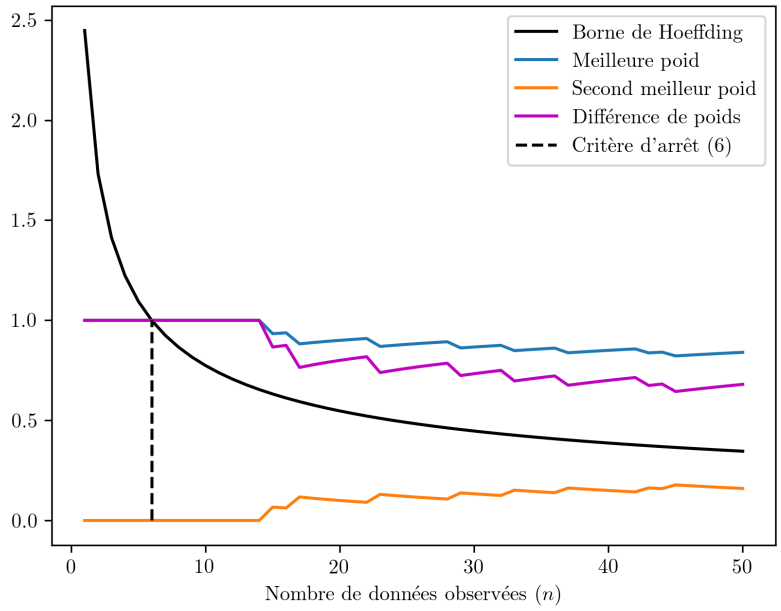


FIGURE 2.4 – Exemple d'application de la borne de Hoeffding lorsque les premières observations donnent toutes la même information. Avec une probabilité de $1 - \delta = 95\%$ et $R = 1$, la borne permet de déterminer le critère d'arrêt à la $6^{\text{ème}}$ observation.

Chapitre 3

Les algorithmes « Very Fast »

3.1 Contexte

Dans le contexte de ce mémoire, l'article *Mining high-speed data streams*[6] est utilisé comme base d'analyse. Cet article présente une modification de l'algorithme des arbres de décision appelée « very fast decision-tree » (VFDT). L'objectif de ce nouvel algorithme est de pouvoir traiter les données provenant du streaming de manière efficace avec un arbre de décision.

Dans un premier temps, ce chapitre introduit les arbres de décision. Ensuite, il explique comment l'algorithme VFDT peut être modifié par l'utilisation de l'inégalité de Hoeffding pour répondre aux problématiques posées par le big data. Une critique mitigeant l'utilisation de l'inégalité est également présentée. Enfin, ce chapitre présente d'autres variantes des arbres de décision utilisant l'inégalité de Hoeffding, ainsi que d'autres algorithmes utilisant l'inégalité pour prendre en compte les spécificités du big data.

3.2 Les arbres de décision

En machine learning, la technique des arbres de décision est une application de la théorie des graphes. Un arbre de décision est un arbre dirigé binaire. Un arbre est un graphe connexe et sans cycle, c'est-à-dire que pour chaque paire de points du graphe, il existe un parcours qui les relie et il n'existe aucun parcours fermé. Le fait de ne pouvoir parcourir les arêtes que dans un sens exprime la partie « dirigé ». Et dans un arbre binaire, un nœud du graphe possède une et une seule arête entrante et seulement deux sor-

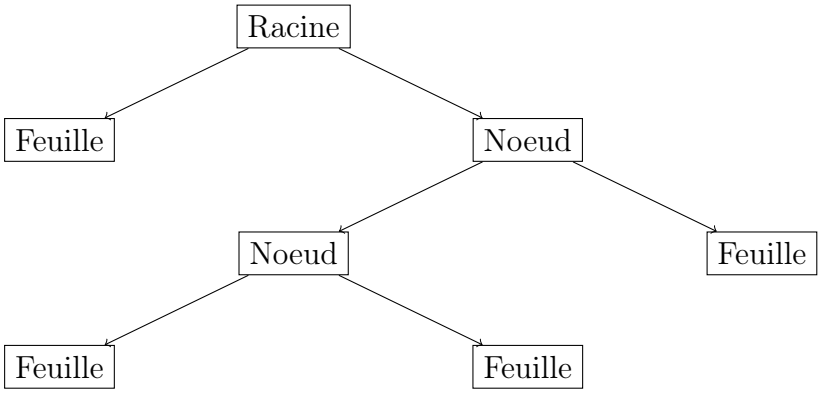


FIGURE 3.1 – Exemple d'arbre binaire dirigé.

tantes. La figure 3.1 montre un exemple d'arbre dirigé binaire. Le premier nœud, ne possédant pas d'arête entrante, est appelé la *racine*. Les nœuds ne possédant pas d'arêtes sortante sont appelés *feuilles*.

Pour faire d'un arbre dirigé binaire un arbre de décision, il faut ajouter des décisions. Pour ce faire, on définit une possibilité de déterminer comment se déplacer dans l'arbre en partant de la racine jusqu'à une feuille. Une fonction dévaluation et une valeur de référence sont ajoutées aux nœuds et à la racine. Ainsi, lorsqu'une donnée sera appliquée à l'arbre de décision, à chaque nœud, la fonction d'évaluation sera évaluée sur la donnée et le résultat donnera l'arête suivante à emprunter. Dans l'exemple de la figure 3.2, lorsque la donnée 37 parcourt l'arbre, les décisions suivantes sont prises :

1. Nœud 1 : $37 \geq 7$ est vrai, l'arête à suivre est donc « vrai » pour atteindre nœud 1
2. Nœud 2 : $37 \geq 77$ est faux, l'arête à suivre est donc « faux » pour atteindre la feuille 4

L'application de cet arbre de décision sur la donnée 37 donne donc le résultat *Feuille 4*.

Pour être plus explicite dans les résultats, un arbre de décision ajoute une information de type *Label* pour nommer les résultats de manière intelligible. Ainsi, la figure 3.3 montre un arbre de décision complet permettant de déterminer si une personne peut ou non lire le journal de Tintin. Dans cet exemple, une personne âgée de 37 ans est autorisée à lire le journal.

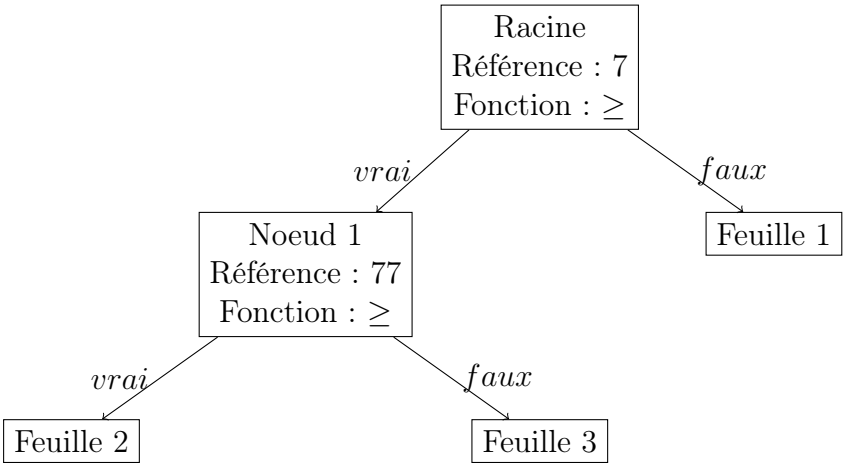


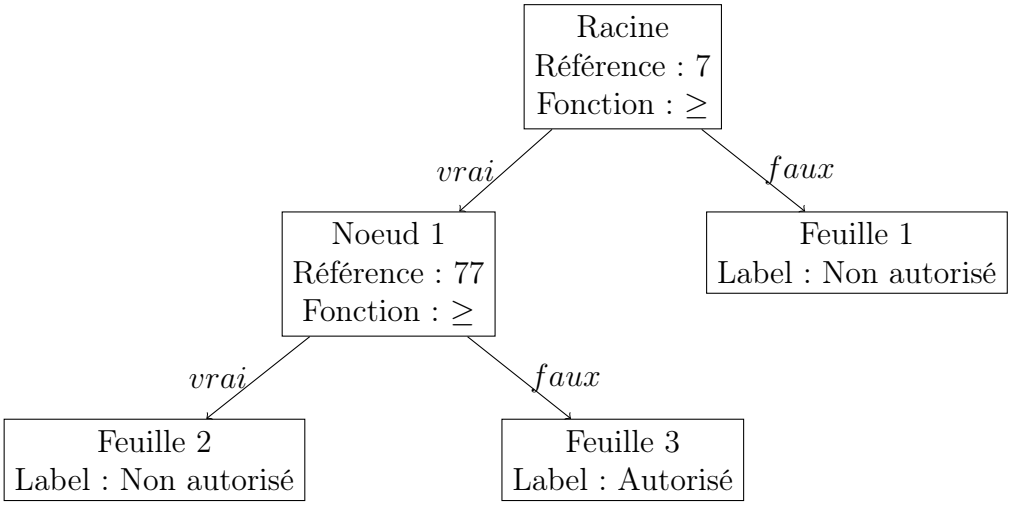
FIGURE 3.2 – Exemple d'arbre binaire dirigé possédant trois feuilles de classification, un nœud de choix et une racine.

La création d'un arbre de décision

Toute technique de machine learning se base sur un modèle qu'il faut construire à partir d'un dataset modèle. Les arbres de décision font partie des techniques dites supervisées qui ont été introduites dans la section §2.1. La manière de créer un modèle est la suivante :

1. Créer la racine et assigner tout le dataset à celle-ci.
2. Sélectionner l'attribut du dataset fournissant la meilleure sous-division grâce à une heuristique. Mémoriser l'attribut utilisé pour ne plus le réutiliser par la suite.
3. Créer deux sous-nœuds et répartir le dataset dans ces deux sous-nœuds en fonction de l'attribut déterminé au point 2.
4. Répéter le processus sur chacun des nouveaux nœuds créés à partir de 2 tant que :
   (a) toutes les données du nœud ne sont pas du même label,
   (b) il reste au moins un attribut sélectionnable,
   (c) il y a des données dans le nœud.

Le choix de l'heuristique à appliquer pour sélectionner le bon attribut est important. La figure 3.4 reprend l'exemple du journal de Tintin mais avec un modèle différent du premier proposé. Bien que ce nouveau modèle donne



1. Référence : Point de comparaison à donner à la fonction d'évaluation avec la donnée actuellement observée
2. Fonction : La fonction d'évaluation à utiliser
3. *vrai/faux* : Les valeurs résultats possibles de la fonction d'évaluation
4. Label : Les labels correspondant au résultat de l'application de l'arbre de décision

FIGURE 3.3 – Exemple d'arbre de décision possédant trois feuilles de classification, un nœud de choix et une racine, représentant l'expression « de 7 à 77 ans »

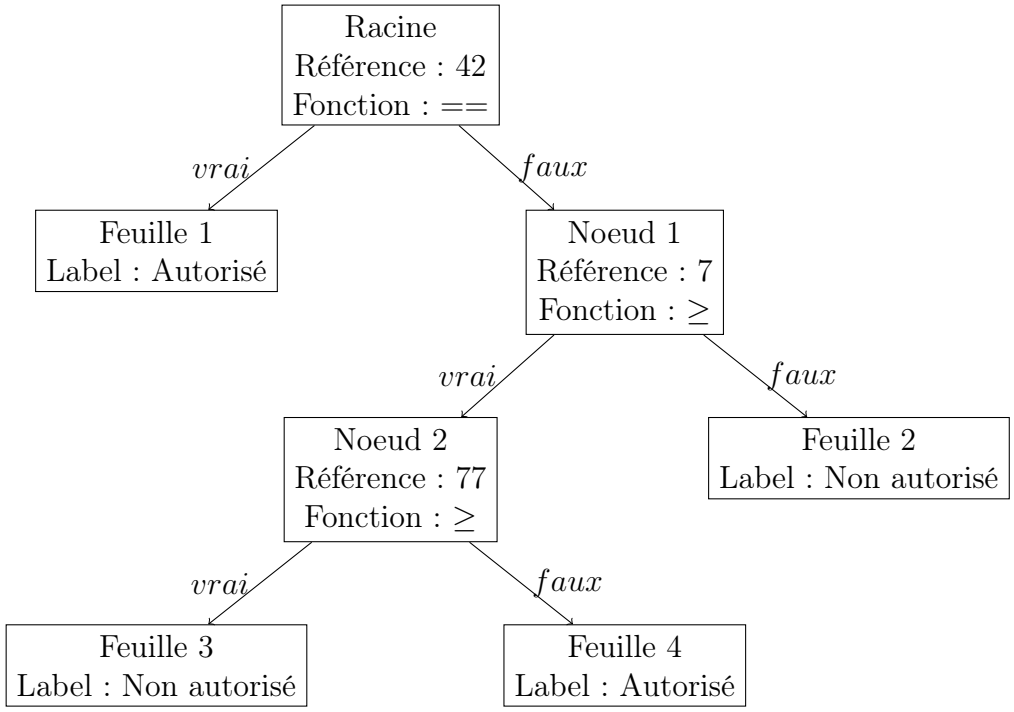


FIGURE 3.4 – Exemple d'arbre de décision possédant quatre feuilles de classification, deux nœuds de choix et une racine, représentant l'expression « de 7 à 77 ans ».

exactement les mêmes résultats que le précédent, celui-ci est plus complexe, ce genre de différence peut être observée selon l'heuristique choisie.

Les heuristiques possibles se basent souvent sur une des deux mesures, l'impureté de Gini et le gain d'information. La première représente la probabilité d'une donnée soit incorrectement labellisée dans un système et doit donc être minimisée. La seconde représente la quantité d'information qu'apporte une donnée dans un système et doit donc être maximisée.

L'overfitting

Une fois l'arbre de décision créé, il se peut que l'application de celui-ci sur les données n'appartenant pas au dataset utilisé pour créer le modèle ne puissent pas être classifiées correctement. En pratique, il arrive que l'arbre de décision créé soit tellement précis et spécialisé pour les données du dataset modèle qu'il soit inutilisable sur d'autres données. Ce phénomène s'appelle

l'*overfitting*. Cette situation peut être détectée grâce aux tests faits lors de l'apprentissage. Ces tests consistent à apprendre seulement une sous-partie du dataset et tenter de classifier le complément une fois la sous-partie apprise. Ainsi, le modèle est validé sur des données n'ayant pas été prises en compte lors de l'apprentissage. Plusieurs techniques existent pour ce faire. La plus simple est de diviser le dataset en une sous-partie et son complément et de faire le test une fois. Cette technique a l'avantage d'être simple et rapide, mais il est possible que le test soit en fait un cas particulier. Une technique plus robuste mais couteuse en temps est de répéter la technique simple plusieurs fois et de faire des statistiques sur les résultats, elle est appelée la *cross validation*. La troisième technique, la plus fiable et moins couteuse que la précédente en terme de temps de calcul est la *k-Fold validation* qui découpe le dataset en k sous-partie. $k-1$ sous-partie sont utilisés pour l'apprentissage et une sous-partie pour la validation. L'apprentissage et la validation sont répétés k fois, avec à chaque fois une sous-partie de validation différente.

Lorsque de l'overfitting est détecté, il faut recréer un autre arbre avec d'autres heuristiques. Lorsque l'arbre de décision est créé, et qu'il n'y a pas overfitting des données, il est alors possible de l'utiliser pour classifier des données du même domaine d'application que celui du dataset modèle.

Les arbres de décision et le big data

Les arbres de décision peuvent être utilisés en combinaison avec le big data. La question qui se pose est : est-ce que les arbres de décision se comportent bien face au big data ? Pour y répondre, il faut reprendre les caractéristiques du big data, les Vs. La description des Vs applicables est décomposée en deux parties, l'une pour l'apprentissage, l'autre pour l'application du modèle à de nouvelles données.

Volume

Apprentissage. Lors de l'apprentissage, le modèle se crée avec un dataset. Dans le cas présent, ce dataset peut contenir un très grand nombre de données. Étant donné que l'algorithme doit stocker les données à chaque nœud pour pouvoir appliquer l'heuristique, la quantité de mémoire nécessaire pour créer un arbre de décision à partir d'un grand nombre de données risque de poser problème.

Application. L'application de l'arbre de décision sur un grand nombre de données n'est pas un problème en soi. Une fois l'arbre créé, les données d'apprentissage peuvent être ignorées et il n'y a pas besoin de stocker les nouvelles données, donc il n'y a pas de souci.

Variety

Apprentissage. La structure d'un arbre de décision est construite autour d'un dataset modèle et d'un type de donnée bien défini, et se base sur les attributs contenus dans celui-ci. Une fois créé, les attributs utilisés pour parcourir l'arbre sont définis et ne peuvent plus changer. Ainsi, une fois construit, un arbre donné ne peut traiter que le type de donnée utilisé lors de la construction.

Application. L'apprentissage figeant le type de donnée utilisable par le modèle, l'application de celui-ci ne peut se faire que sur ce type de donnée également. De plus, même avec un type de donnée ne changeant pas, sir le domaine des données change, en d'autre termes si y a un drift sur les données, un arbre de décision n'en prend pas compte.

Velocity

Apprentissage. Lors de l'apprentissage, à chaque fois qu'un nœud est créé, il faut recalculer les heuristiques, et donc repasser sur une grande partie des données encore et encore. Le temps d'apprentissage peut devenir extrêmement long à cause de cela.

Application. Le temps d'application du modèle à une donnée est le temps que cette donnée passe dans les nœuds jusqu'à atteindre une feuille. Ce temps est directement dépendant de la manière dont l'arbre est structuré. En réalité, l'application du modèle répond exactement aux même critères qu'un arbre de recherche binaire de la théorie des graphes. Or, cette théorie dit que le temps de recherche dans un arbre dépend directement de la taille du chemin racine-feuille le plus long qui existe dans l'arbre. Dans le pire des cas, l'arbre est totalement déséquilibré, et le chemin le plus long a un longueur égale au nombre de nœuds dans l'arbre. Ainsi, la recherche est au pire d'ordre $\mathcal{O}(n)$ avec n le nombre de nœuds dans l'arbre. Dans le meilleur des cas, l'arbre est équilibré, et alors tous les chemins ont la même longueur ($^+/_-1$). Dans

ce cas, On peut montrer que cette longueur est $log(n)$ avec n le nombre de nœuds dans l'arbre, et donc la recherche au mieux est d'ordre$\mathcal{O}(log(n))$. Avec cette observation, il est clair que la structure créée lors de l'apprentissage va conditionner fortement le comportement de l'arbre vis à vis de la vélocité.

Veracity

Apprentissage. En fonction de l'heuristique choisie pour l'apprentissage, les données parasites auront ou non un impact. Néanmoins, toutes les heuristiques travaillent sur des statistiques liées aux données et ne supposent pas qu'elles soient parfaites. Le risque d'une donnée parasite est qu'elle entraine une tendance à l'overfitting.

Application. Lors de l'application du modèle sur les données, le fait qu'une donnée parasite soit classifiée n'est pas vraiment un problème. Les arbres de décision, comme beaucoup d'autres algorithmes en machine learning, ne sont pas fiables à 100%. Il y a toujours une légère erreur possible dans les classifications, une donnée parasite sera très probablement intégrée à cette erreur. Les données parasites sont en fait assimilées à du bruit.

3.3 Very fast decision tree (VFDT)

Un des points limitant des arbres de décision dans le cadre du big data est que lors de l'apprentissage, ils doivent stocker les données observées et les ré-observer à chaque fois qu'un nouveau nœud est créé. L'algorithme du very fast decision tree[6] aborde cette problématique.

L'idée est d'observer les données du dataset modèle une à une lors de l'apprentissage et de les compter. Ce compte est alors utilisé par l'heuristique d'évaluation. La donnée en elle-même peut être oubliée définitivement. Les données comptées dans un nœud donné ne sont plus utilisées dans les nœuds suivants. Cela est possible car dans le cadre du big data, il y a énormément de données, et donc les nœuds suivants seront peuplés par la suite des données contenues dans le dataset modèle.

L'heuristique utilisée permet également de limiter la création de nœuds, ce qui permet de réduire la taille de l'arbre et donc d'améliorer ses performances lors de son application. Dans les arbres de décision, l'heuristique permet de

connaitre quel attribut utiliser pour créer un nouveau nœud. Dans le VFDT, l'heuristique doit également dire s'il faut créer un nœud ou pas.

L'heuristique choisie se base sur l'inégalité de Hoeffding.

L'algorithme

L'algorithme 3.5 montre l'algorithme original[6]. Voici une retranscription textuelle montrant les étapes principales de l'algorithme :

- Initialisation
  - Création d'un nœud vide.
  - Initialisation à 0 des éléments d'un tableau (T_{la}) à deux dimensions, l'une étant le nombre de labels différents gérés par l'arbre de décision final, l'autre étant le nombre d'attributs liés aux données.
  - Initialisation à 0 des éléments d'un tableau (T_l) à une dimension de taille égale au nombre de labels différents gérés par l'arbre de décision final.
- Prise en compte d'une donnée du dataset modèle
  - Appliquer l'arbre de décision en cours de création sur la donnée afin de déterminer la feuille correspondante.
  - Incrémentation du tableau T_l à l'index correspondant au label de la donnée.
  - Assignation du label majoritaire dans la feuille courante.
  - Si la feuille contient plus d'un label :
    - Calcul d'une évaluation (G) de chaque attribut en utilisant T_{la}.
    - Calcul de la borne de Hoeffding (ϵ) avec n le nombre total de données déjà observées dans cette feuille en se basant sur T_{la}.
    - Récupération des deux meilleurs évaluations de G (G_1 et G_2).
    - Si $|G_1 - G_2| > \epsilon$, alors la feuille courante est transformée en nœud et deux nouvelles feuilles sont créées. Le nouveau nœud utilise l'attribut lié à G_1 pour décider si une donnée doit aller dans la première nouvelle feuille ou la seconde.

La création de l'arbre se termine lorsque toutes les données du dataset ont été prises en compte.

Les fonctions d'évaluations utilisables sont les mêmes que celles citées à la section §3.2. Leur utilisation pose toutefois problème à cause des conditions qu'apporte l'inégalité de Hoeffding. Ce point est débattu dans la section §3.4.

Inputs: S is a sequence of examples,
$\mathbf{X}$ is a set of discrete attributes,
$G(.)$ is a split evaluation function,
δ is one minus the desired probability of choosing the correct attribute at any given node.
Output: HT is a decision tree.

Procedure HoeffdingTree $(S, \mathbf{X}, G, \delta)$
Let HT be a tree with a single leaf l_1 (the root).
Let $\mathbf{X_1} = \mathbf{X} \cup \{X_\emptyset\}$.
Let $\overline{G}_1(X_\emptyset)$ be the $\overline{G}$ obtained by predicting the most frequent class in S.
For each class y_k
  For each value x_{ij} of each attribute $X_i \in \mathbf{X}$
    Let $n_{ijk}(l_1) = 0$.
For each example $(\mathbf{x}, y_k)$ in S
  Sort $(\mathbf{x}, y)$ into a leaf l using HT.
  For each x_{ij} in $\mathbf{x}$ such that $X_i \in \mathbf{X}_l$
    Increment $n_{ijk}(l)$.
  Label l with the majority class among the examples seen so far at l.
  If the examples seen so far at l are not all of the same class, then
    Compute $\overline{G}_l(X_i)$ for each attribute $X_i \in \mathbf{X}_l - \{X_\emptyset\}$ using the counts $n_{ijk}(l)$.
    Let X_a be the attribute with highest $\overline{G}_l$.
    Let X_b be the attribute with second-highest $\overline{G}_l$.
    Compute ϵ using Equation 1.
    If $\overline{G}_l(X_a) - \overline{G}_l(X_b) > \epsilon$ and $X_a \neq X_\emptyset$, then
      Replace l by an internal node that splits on X_a.
      For each branch of the split
        Add a new leaf l_m, and let $\mathbf{X_m} = \mathbf{X} - \{X_a\}$.
        Let $\overline{G}_m(X_\emptyset)$ be the $\overline{G}$ obtained by predicting the most frequent class at l_m.
        For each class y_k and each value x_{ij} of each attribute $X_i \in \mathbf{X_m} - \{X_\emptyset\}$
          Let $n_{ijk}(l_m) = 0$.
Return HT.

FIGURE 3.5 – Procédure de création d'un very fast decision tree. Reprise de l'article *Mining High-speed Data Streams*[6]

Il est intéressant de remarquer également que l'arbre ne grandit que lorsque $G_1 - G_2 > \epsilon$. Ceci permet de limiter l'accroissement de l'arbre, car cette inégalité demande à observer plusieurs éléments avant d'être vérifiée. Cette comparaison permet aussi d'assurer à une probabilité $1 - \delta$ près que G_1 est suffisamment différencié de G_2 lors du choix de G_1 comme paramètre de séparation. Ceci rend l'arbre de décision ainsi créé théoriquement meilleur que celui créé par l'algorithme classique.

Le concept de drift n'est toujours pas pris en charge par le nouvel algorithme. La section §3.5 discute une amélioration du VFDT pour prendre en compte ce point.

Le VFDT et le big data

Le VFDT permet de répondre à certains des défis posés par le big data. Ce nouvel algorithme modifie la création de l'arbre de décision, pas son application une fois créé. Ainsi, les limitations liées à l'application de l'arbre présentées dans la section 3.2 restent d'actualité. Voici en résumé les améliorations qu'apporte le VFDT par rapport aux 5Vs :

Volume

L'algorithme classique de la création d'arbre de décision a besoin de stocker toutes les données observées lors de l'apprentissage pour pouvoir prendre la décision de créer de nouvelles feuilles ou pas. L'algorithme du VFDT ne stocke pas les données, il ne stocke que deux tableaux dans les feuilles. Le premier a une taille égale au nombre de labels présents dans le dataset modèle multiplié par le nombre d'attributs contenus dans les données. Le second a une taille égale au nombre de labels présents dans le dataset modèle. Ces tailles sont ridiculement petites par rapport à la quantité de données présentes dans le dataset. Pour donner un point de comparaison, l'article présentant le VFDT travaille sur des dataset modèles comportant de 1000 à $1e^{+8}$ données. Les datasets utilisés plus tard dans ce document comportent au maximum 34 attributs et 10 labels, ce qui donne au maximum 350 valeurs numériques à stocker par feuille. Le souci de place mémoire est donc maîtrisé.

Variety

L'algorithme proposé a les mêmes limitations que celui des arbres de décision classique.

Velocity

Lors de l'apprentissage, étant donné que les données ne sont plus stockées mais transformées en un ensemble limité de valeurs, le temps pour calculer les heuristiques est fortement réduit. De plus, l'inégalité de Hoeffding ayant besoin d'un minimum d'échantillons pour commencer à pouvoir être utilisée comme expliqué dans la section §2.3, il n'y a même pas besoin de calculer les heuristiques tant que ce nombre d'échantillons n'a pas été atteint.

Grâce à l'inégalité de Hoeffding, l'arbre de décision créé est plus petit qu'un arbre classique créé sur les mêmes données, et donc la taille des chemins allant de la racine aux feuilles est en moyenne plus petite. Ceci induit un temps de recherche plus court lors de l'application du modèle à une nouvelle donnée.

Veracity

L'inégalité de Hoeffding compare deux valeurs heuristiques de deux attributs distincts G_1 et G_2 et assure à une probabilité $1 - \delta$ près que G_1 est bien la valeur à utiliser pour créer les nouvelles feuilles. Ainsi, la création de l'arbre de décision est beaucoup plus robuste aux imperfections des données car ces imperfections sont lissées grâce à ce mécanisme. Au pire, si les imperfections sont nombreuses, l'algorithme attendra plus d'échantillons avant de décider qu'il faut créer de nouvelles feuilles.

3.4 Limitations de l'inégalité de Hoeffding

Comme toute formule mathématique, l'inégalité de Hoeffding ne s'applique que sous certaines conditions. L'article introduisant le VFDT[6] ainsi que les articles traitant des dérivés du VFDT[19, 7, 11, 12] ne vérifient malheureusement pas ces conditions. D'après *L. Rutkowski, L. Pietruczuk, P. Duda and M. Jaworsk*[18] ces conditions ne sont en fait pas respectées, et l'article propose une autre inégalité pour résoudre ce problème.

Les conditions d'utilisations de l'inégalité de Hoeffding suivantes ne sont pas respectées dans les algorithmes liés au VFDT[18] :

Sommes

L'inégalité de Hoeffding compare la probabilité (Pr) que deux valeurs (S, $E[S]$) diffèrent de plus d'ϵ par rapport à une valeur calculée, appelée la borne de Hoeffding. Les deux valeurs comparées sont des sommes, l'une étant la somme des variables aléatoires indépendantes et l'autre la valeur réelle attendue de la variable multipliée par le nombre de variables dans la somme

$$Pr(S - E[S] \geq \epsilon) \leq borne \tag{3.1}$$

$$S = \sum_{i=1}^{N} Y_i \tag{3.2}$$

$$E[S] = N.E[Y] \tag{3.3}$$

où $E[Y]$ est la valeur réelle attendue pour la variable. Or, le gain d'information et l'index de Gini utilisés dans les algorithmes comme heuristiques ne peuvent pas être exprimés sous forme de sommes. Ceci rend l'utilisation de la borne de Hoeffding impossible dans ces cas précis.

Valeurs continues

L'inégalité de Hoeffding suppose que les données sont des données continues. Ceci n'est pas le cas dans l'expression générale des algorithmes.

L'article[18] mettant ces faits en évidence propose l'utilisation de l'inégalité de McDiarmid[15] en lieu et place de l'inégalité de Hoeffding, la seconde étant un cas particulier de la première. L'article prouve que les conditions non respectées ne sont pas d'application avec cette nouvelle inégalité. Il introduit également les formules ad-hoc pour prendre en compte le gain d'information ainsi que l'index de Gini correctement.

Par contre, les expériences pratiques montrent que l'utilisation de l'inégalité de McDiarmid tend à générer des arbres de décision moins précis que ceux créés par l'inégalité de Hoeffding. Cela s'explique par le fait que l'inégalité de McDiarmid est plus dure à satisfaire, et donc à δ égal, celle-ci demandera plus d'éléments à observer avant de permettre la création de nouvelles feuilles que l'inégalité de Hoeffding. C'est à tel point que les arbres de décision générés sont trop petits par rapport à la complexité des recherches à faire ce

qui entraine un phénomène d'*underfitting*.

Pour éviter cet underfitting, l'article suggère, pour un travail futur, d'ajouter une condition supplémentaire qui forcerait la création de nouvelles feuilles lorsque la borne de McDiarmid est trop petite.

Il est intéressant de remarquer sur ce dernier point que l'implémentation C proposée par Goeff Hulten, l'auteur du VFDT, introduit également une condition de ce genre sur la borne de Hoeffding. L'explication donnée est exactement la même que celle pour la borne de McDiarmid : éviter l'underfitting.

3.5 Concept-adapting very fast decision tree

Comme discuté dans la 3.3, les VFDT se comportent assez bien face au big data. Mais un phénomène n'est pas pris en compte du tout : le *drift*. Le drift exprime le fait que les données observées à un temps T_0 peuvent changer de signification à un temps T_i ultérieur. Le VFDT considère que les observations se font toujours sur un domaine figé. Or, dans le big data, étant donné que les données arrivent en continu et sur le long terme, il est très probable que le domaine d'application de ces données change avec le temps. Il suffit de regarder comment étaient utilisés les réseaux sociaux à leur début et maintenant : le temps fait changer les choses.

Le souci de ce phénomène est qu'un arbre de décision précis et efficace créé au temps T_0 peut très bien perdre sa précision et ne plus être capable de classifier correctement au temps T_i. Les créateurs du VFDT introduisent le *concept-adapting very fast decision tree* [12](CFDT), une modification du VFDT pour prendre le phénomène de drift en compte. Pour rappel, chaque nœud d'un VFDT contient une valeur de référence et une fonction de comparaison permettant de diriger une donnée observée vers un de ses deux sous-nœuds. Cette valeur est déterminée par une heuristique et l'inégalité de Hoeffding.

Le principe de la modification est de vérifier à chaque nœud que la donnée observée ne modifierait pas les choix qui ont été faits lors de la détermination de la valeur de référence. Si c'est le cas, alors un nouveau nœud est créé et un nouveau sous arbre est créé petit à petit. A chaque observation, les précisions des deux nœuds sont comparées, l'ancien nœud est remplacé lorsque le nouveau devient plus précis que lui. D'un point de vue des contraintes de volume du big data, la mémoire utilisée est supérieure à celle du VFDT mais

toujours constante par rapport au nombre d'élément observés. Cette modification n'impact pas les autres Vs. D'un point de vue précision, le CVFDT est aussi précis que le VFDT sur des domaines ne changeant pas. Par contre, il prend bien en compte les changements de modèle et devient meilleur avec le temps.

3.6 Very fast machine learning

Very fast k-Means

L'algorithme du k-Means est, contrairement aux arbres de décision, une technique non-supervisée et permet de regrouper différentes données entre elles. Cet algorithme est un algorithme itératif, il détermine les groupes de plus en plus précisément en fonction de l'itération précédente. L'algorithme s'arrête lorsqu'il considère avoir trouvé les bons groupes. Il existe différentes techniques pour arriver à un résultat, elles peuvent agir sur la manière de déterminer les groupes la première fois, sur la manière de calculer les groupes par la suite ou sur la manière de déterminer le moment où les bons groupes sont trouvés.

Pedro Domingos et Geoff Hulten[19] proposent une méthode permettant de réduire le temps d'apprentissage en limitant le nombre d'éléments du dataset modèle à observer. En d'autres termes, lors du calcul des groupes, au lieu d'utiliser toutes les données présentent dans le dataset d'apprentissage, l'algorithme va observer une sous partie. L'inégalité de Hoeffding est utilisée pour estimer les erreurs de position des centroïdes trouvés grâce au sous ensemble par rapport aux positions qui auraient été trouvées en utilisant toutes les données. Ces erreurs sont alors minimisées d'itération en itération.

Very fast EM

L'algorithme de Expectation-maximisation[4] est, comme le k-Means, une manière de regrouper différentes données entre elles. Cet algorithme est également itératif et détermine les groupes en alternant une phase d'évaluation et une phase de maximisation, chacune de ces phases se basant sur les résultats précédents de l'autre pour s'exécuter.

Comme le k-Means, le EM doit observer les données pour pouvoir déterminer les groupes. Pedro Domingos et Geoff Hulten[7] généralisent la méthode

proposée lors de leurs recherche sur le VFKM[19] et l'applique à l'EM afin de réduire la quantité de données à devoir observer pour arriver au résultat attendu.

Implémentation de référence

Pour faciliter l'implémentation de tous ces algorithmes, Pedro Domingos et Geoff Hulten ont créé un framework baptisé VFML[11] qui encapsule un ensemble de primitives réutilisables. Ce framework est écrit en C et est disponible en Open Source sur le site de l'université de Washington (http ://www.cs.washington.edu/dm/vfml/).

Pour des raisons de facilité de prototypage et d'outils, les algorithmes développés dans le cadre de ce mémoire l'ont été en Python, ils n'utilisent donc pas les primitives proposées dans le VFML.

Chapitre 4

Les plus proches voisins

4.1 Contexte

Dans le cadre de ce mémoire, l'article présentant le VFDT[6] a été étudié ainsi que certains de ses dérivés. Il a été mis en évidence que pour répondre aux défis posés par le big data sur les arbres de décision, une approche utilisant l'inégalité de Hoeffding permettait de répondre à certains de ces défis.

Dans un second temps, ce mémoire se penche sur une autre technique utilisée dans le machine learning à laquelle l'inégalité de Hoeffding peut être appliquée. Parmi un ensemble important de techniques diverses et variées, la technique des plus proches voisins a été choisie. Cette technique est assez simple à comprendre et à implémenter, ce qui permet facilement de la modifier pour y introduire l'inégalité de Hoeffding. De plus, l'inégalité de Hoeffding est fonction d'un nombre d'éléments n (cf. équation (2.3)) et les plus proches voisins se basent sur une recherche de k éléments voisins de l'élément observé. La description de l'algorithme des plus proches voisins se fait dans la section §4.2, mais même sans celle-ci, intuitivement, on peut associer le n de l'inégalité de Hoeffding et le k des plus proches voisins. La suite de ce document montrera comment, en pratique, ce lien est fait.

Dans la suite, le chapitre 5 propose une modification de l'algorithme des plus proches voisins qui permet de rendre le nombre de voisins à prendre en compte lors de la classification dépendant de la position de l'élément à classifier. Ensuite, le chapitre 7 propose une technique pour réduire la quantité de données fournies à l'algorithme des plus proches voisins tout en gardant un maximum de précision.

4.2 Le kNN

L'algorithme des plus proches voisins (*k-nearest neighbors* abrégé en kNN) permet la classification d'éléments en se basant sur une représentation spatiale des données. La classification permet de déterminer avec une certaine probabilité si un élément observé doit être labellisé avec un label ou un autre. Plus précisément, à chaque donnée observée, le kNN peut donner la probabilité que la donnée soit d'un label présent dans le système et ce pour chaque label.

Le kNN se base sur une représentation spatiale du dataset modèle. Cet espace possède autant de dimensions que d'attributs du dataset. Les données du dataset et celles observées doivent posséder les mêmes attributs pour que l'algorithme fonctionne correctement. Certaines techniques permettent de pallier à l'absence de l'un ou l'autre de ces attributs, comme la suppression de l'attribut dans toutes les données (réduction de la dimension de l'espace), l'assignation d'une valeur unique à la donnée, l'assignation d'une valeur moyenne à la donnée, etc... Mais ces techniques ne sont pas abordées plus avant dans ce mémoire. Les algorithmes proposés dans la suite supposent que les données n'ont pas d'attribut manquant, tout comme le fait l'implémentation de référence utilisée dans le cadre de ce mémoire[17].

Le kNN est un algorithme faisant partie du groupe de l'apprentissage supervisé. Ainsi, un dataset modèle doit être fourni et appris. L'apprentissage basique du kNN consiste à stocker le dataset de manière à pouvoir le parcourir lors de la classification d'une nouvelle donnée, et à déterminer le nombre de voisins (k) qu'il faudra utiliser lors de la classification d'une nouvelle donnée. L'apprentissage est décrit en plus de détails dans la *section 4.2.2.*

La classification par le kNN consiste à trouver les k plus proches voisins de la donnée observée dans l'espace multidimensionnel et de compter la proportion de chaque label assigné à ces voisins. Le label à assigner à la donnée observée est le label avec la plus grande proportion. Le tableau 4.1 et la figure 4.1 montrent un exemple en 2 dimensions. Ceux-ci donnent le dataset modèle, les 4 plus proches voisins du point $(.5, .5)$, la probabilité de chaque label, et enfin le label choisi pour le point observé. A_x et $A_y$sont les attributs des données, ici ce sont des coordonnées dans un plan. L_1, L_2, L_3et L_4 sont les labels possibles présent dans le dataset. Pour le point $(.5, .5)$, le label trouvé est L_1 avec une probabilité de 50%.

Dataset		
A_x	A_y	Label
0	0	L_1
1	0	L_1
0	1	L_2
1	1	L_3
-1	0	L_4
0	-1	L_4

k plus proches voisins de $(.5, .5)$		
A_x	A_y	Label
0	0	L_1
1	0	L_1
0	1	L_2
1	1	L_3

Résultats pour $(.5, .5)$		
Labels	Probablités	Résultat
L_1	50%	L_1
L_2	25%	
L_3	25%	
L_4	0%	

TABLE 4.1 – Exemple de classification des plus proches voisins sur la donnée $(A_x = .5, A_y = .5)$ avec $k = 4$.

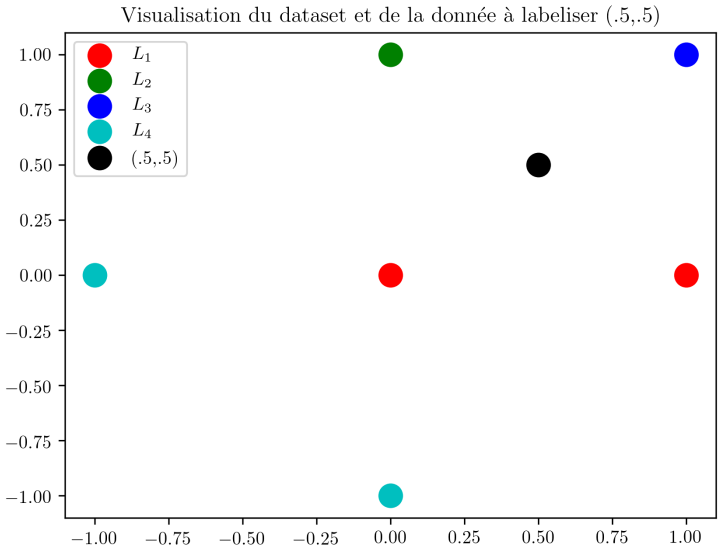


FIGURE 4.1 – Exemple de classification des plus proches voisins sur la donnée $(A_x = .5, A_y = .5)$.

4.2.1 Application du kNN

L'application du kNN à une nouvelle donnée ne semble pas complexe ; il suffit de trouver les k plus proches voisins dans l'espace et de compter leur label respectifs comme le montre le tableau 4.1.

Néanmoins, la procédure *SearchNeighbors* cache un parcours du dataset modèle X_m. En effet, cette procédure doit calculer la distance de chacun des points de $X_m$et les trier pour pouvoir en déduire les k plus proches de X_c. Cette approche simple est appelée l'approche *brute force*. Elle est fonctionnelle, mais peut prendre beaucoup de temps sur des datasets modèle important en calculant toutes les distances de manière exhaustive. La section 4.2.2 expose deux autres mécanismes permettant de réduire ce temps perdu, et donc d'améliorer les performances de l'application du kNN. Ces techniques observent les données lors de l'apprentissage et les décorent de données utilisées par la suite dans la procédure *SearchNeighbors* pour réduire le nombre de distances à calculer.

La procédure *SearchNeighbors* doit calculer la distance des points au point observé. Plusieurs calculs de distances existent dans la littérature, certaines plus ou moins indiquées en fonction de la manière dont les données ont été

Algorithme 4.1 Procédure pour la classification d'un élément en se basant sur l'algorithme kNN.

X_m	un modèle
X_c	un élément à classifier
k	le nombre de plus proche voisins à prendre en compte

Procedure *Classify*
  neighbors=*SearchNeighbors*(X_m, X_c, k)
  foreach neighbor **in** neighbors
    neighborClass = *GetClass*(neighbor)
    accumulator[neighborClass]++
  return *MaxClass*(accumulator)

SearchNeighbors	Retourne les k plus proches voisins de X_c dans X_m
GetClass	Retourne la classe de *neighbor* en fonction de X_m
MaxClass	Retourne la classe maximale dans *accumulator*

segmentées comme discuté dans la section *Détermination de la technique de recherche*. Néanmoins, en pratique la distance Euclidienne est presque toujours utilisée.

4.2.2 Apprentissage du kNN

Détermination du k

Le point obligatoire lors de l'apprentissage du kNN est la détermination d'un k qui sera utilisé dans la procédure $SearchNeighbors$. L'algorithme 4.2 montre la logique utilisée pour la détermination du k. Cet algorithme est très proche de celui utilisé lors de la validation des tests, il est dérivé de la *k-Fold Validation*. Cette procédure peut prendre un certain temps mais n'est à faire qu'une seule fois.

Les trois datasets présentés dans la suite de ce document sont générés automatiquement, et permettent de voir quelles valeurs peut prendre le k et d'analyser celles-ci.

Le premier dataset est exposé figure 4.2 où les classes du modèle sont fortement distinctes. Dans cet exemple, un k de 1 est suffisant pour obtenir une précision parfaite (100%). L'espace de cet exemple se divise facilement en deux parties bien distinctes et le kNN n'a pas besoin d'aller voir beaucoup de voisins pour faire une prédiction valable. Bien sûr, ce genre de répartition n'arrive jamais dans les faits. Le plus souvent, les données se chevauchent au

Algorithme 4.2 Logique utilisée pour déterminer k lors de l'apprentissage du kNN. Inspiré de l'algorithme de validation *k-Fold*.

X_m	un modèle
n	Le nombre de sous-divisions à faire

```
Procedure DefineK
  ks = []
  X_mn = SplitDataset(X_m, n)
  foreach X_mi in X_mn
    bestK = 0
    bestScore = 0
    X_mtrain = X_m − X_mi
    foreach k in [1, Size(X_mtrain)]
      score = Score(X_mtrain, X_mi, k)
      if bestScore < score
        Add(ks, k)
        bestScore = score
  return ⌈Mean(ks)⌉
```

SplitDataset	divise le dataset $X_m$en k sous-parties de taille identique
Score	calcul la précision du kNN sur X_{mtrain}avec les données X_{mi}et en regardant les k plus proches voisins
Add	Ajoute une donnée au tableau donné
Mean	retourne la moyenne du tableau donné

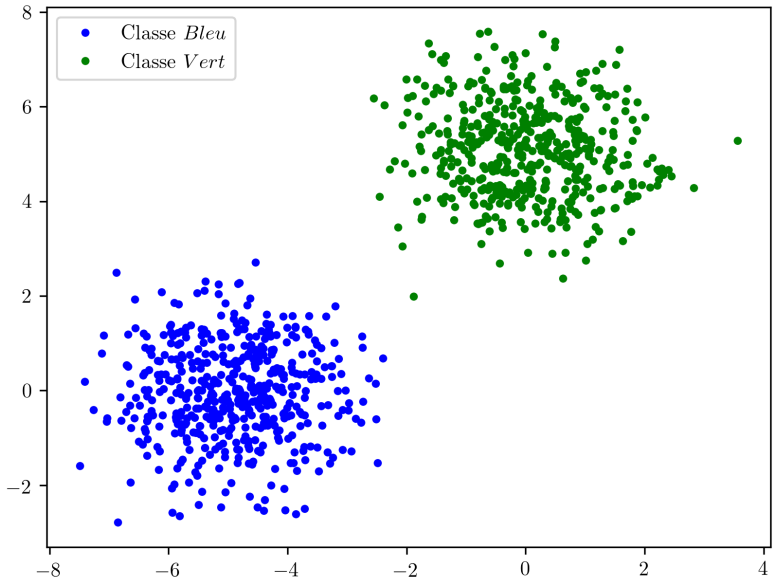


FIGURE 4.2 – Exemple de modèle pour le kNN contenant deux classes bien distinctes.

moins légèrement.

Le second dataset est exposé figure 4.3 où les classes du modèle se recouvrent fortement. Dans cet exemple, un k de 10 est optimal pour une précision de 91%. Cela s'explique par le fait que l'espace de cet exemple est plus difficile à diviser et donc, le nombre de voisins à prendre en compte est plus élevé et les prédictions ne sont pas parfaites dans les zones se recouvrant.

Le troisième dataset est exposé figure 4.4 où deux des classes se recouvrent et une est clairement distincte. Dans cet exemple, un k de 9 est optimal pour une précision de 95%. L'ajout d'une classe indépendante des deux autres ne permet pas de réduire significativement la taille du k, celui-ci reste relativement élevé. En même temps, la précision augmente grâce à la classe distincte dont les données sont majoritairement bien classifiées. Et donc, en moyenne, la précision du modèle est meilleur. Néanmoins, observer 9 voisins dans la partie du modèle isolée semble sous-optimal. L'algorithme du kNN est ainsi fait qu'il n'est pas possible d'avoir plusieurs k en fonction de l'endroit où l'élément à prédire se trouve. Pour adresser ce problème, ce mémoire propose une modification au kNN appelé le kLNN dans le chapitre 5.

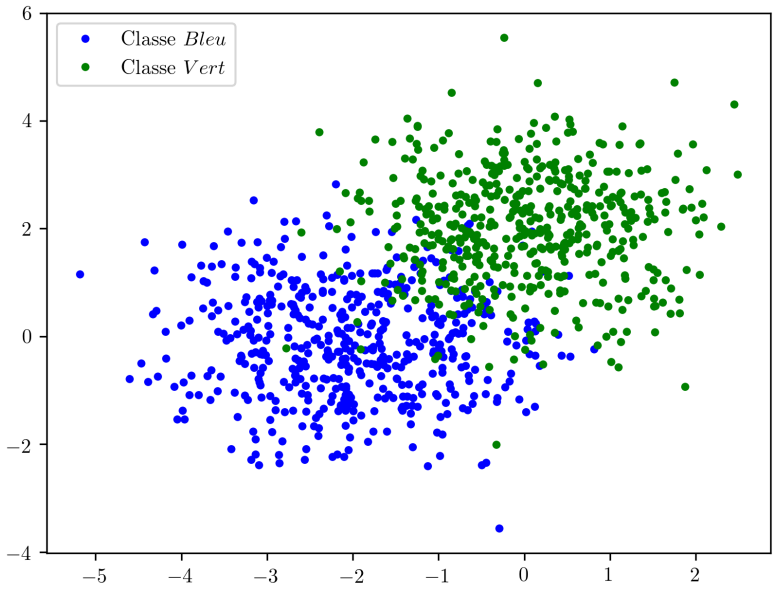


FIGURE 4.3 – Exemple de modèle pour le kNN contenant deux classes se chevauchant.

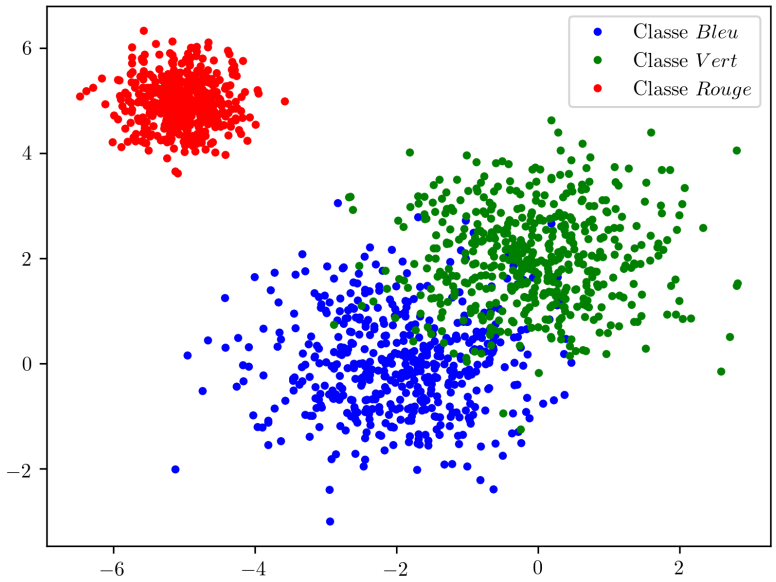


FIGURE 4.4 – Exemple de modèle pour le kNN contenant deux classes se chevauchant et une troisième distincte.

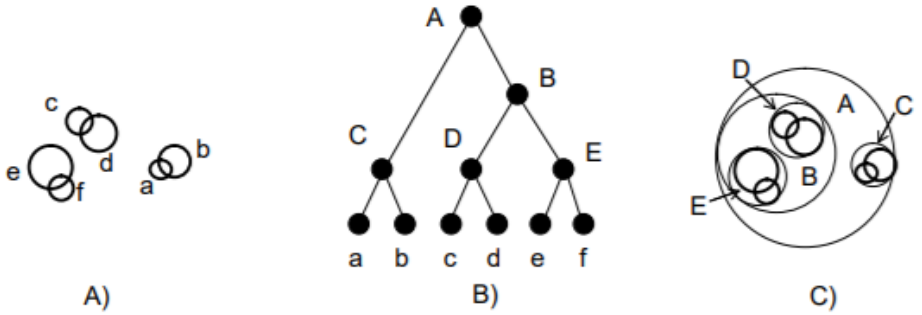


FIGURE 4.5 – Exemple d'un BallTree. Repris de [16]. A) Un ensemble d'hypersphères dans le plan. B) Un BallTree donnant accès à ces hypersphères. C) Les hypersphères résultantes du BallTree.

Détermination de la technique de recherche

Comme exposé dans la section 4.2.1, le dataset modèle du kNN peut être simplement stocké puis parcouru par la procédure *SearchNeighbors* de l'algorithme 4.1, mais il peut aussi être agrémenté d'informations optimisant cette procédure.

Dans le framework utilisé par ce mémoire, le module python *sklearn.neighbors*, propose trois techniques de parcours, le balltree, le kdtree et la force brute :

BallTree. Comme le montre la figure 4.5, la technique du BallTree[16] divise l'espace multidimensionnel en un ensemble d'hypersphères de même dimension imbriquées les unes dans les autres et liées entre elles par un arbre binaire. Une particularité de cette technique est de permettre à ces hypersphères de se chevaucher et de ne pas avoir besoin d'englober l'ensemble de l'espace observé.

KDTree. Comme le montre la figure 4.6, la technique du KDTree[2] divise l'espace en un arbre binaire dont chaque nœud représente un hyperplan divisant l'espace en deux. Ce plan est perpendiculaire à une des dimensions de l'espace, ce qui permet de définir rapidement si un élément de l'espace est d'un côté ou de l'autre par rapport à l'axe choisi. L'article [2] montre que la recherche des plus proches voisins dans un KDTree peut être d'ordre $\mathcal{O}(log(n))$. Il cite également [9] qui apporte une modification du KDTree et qui permet d'atteindre ces performances avec moins de contraintes. L'article

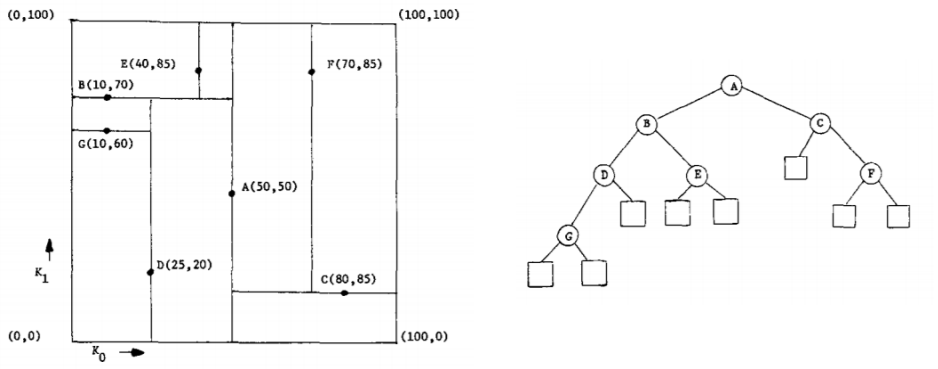


FIGURE 4.6 – Exemple d'un KDTree. Repris de [2]. À gauche, la visualisation des hyperplans en deux dimensions avec les points de références permettant de les identifier. À droite, l'arbre binaire sous-tendant.

[13] montre les différences de performance de plusieurs techniques, dont le BallTree et le KDTree, et conclut que si les données recherchées ont une répartition similaire aux données de l'espace, le KDTree est le meilleur algorithme. Le KDTree est le plus efficace lorsqu'il est utilisé de concert avec la distance de Minkowski ($\sqrt[p]{\sum |x-y|^p}$) avec un p infini [2]. La modification proposée dans l'article [9] permet d'utiliser n'importe quel p, en particulier $p = 2$, ce qui donne la distance Euclidienne.

Brute Force. Cette technique consiste à parcourir tous les points un à un.

Cohérence des données

Le kNN calcule des distances entre des points dans un espace multidimensionnel. Chaque dimension représente un attribut des données du dataset. Les calculs de distances n'étant pas pondérés en fonction de la dimension, un attribut ayant un ordre de grandeur très élevé aura une grande influence par rapport à un attribut ayant un ordre de grandeur inférieur.

Il est donc recommandé de normaliser les données, c'est-à-dire de réechantillonner les attributs pour qu'ils soient tous dans l'intervalle $[0 - 1]$. Ainsi, toutes les données auront le même poids lors du calcul des distances.

Chapitre 5

Le k local nearest neighbors (kLNN)

L'algorithme des plus proches voisins a été sélectionné comme base de réflexion dans le cadre de l'application de l'inégalité de Hoeffding sur un autre algorithme que les arbres de décision. Ce choix a été fait après avoir pris en compte un ensemble d'algorithmes liés au machine learning car, il est relativement simple et rapide à ré-implémenter le cas échéant. Une réflexion a alors été menée pour tenter de trouver comment utiliser l'inégalité de Hoeffding.

L'idée principale sous-tendant la réflexion est de tenter de lier le nombre de voisins à observer pour classifier une donnée (le k du kNN) et le nombre d'éléments dans les sommes nécessaire à l'inégalité de Hoeffding (le n de l'équation (2.3)). La réflexion se simplifie alors à « comment déterminer le k optimum avec l'inégalité de Hoeffding ? ».

Pour aider à la réflexion, un dataset synthétique a été créé, visible sur la figure 5.1. Les données de celui-ci contiennent deux attributs et les labels possibles sont : *Rouge*, $Vert$, *Bleu*. Les données des trois labels sont réparties autour d'un point, différent pour chaque label, de manière gaussienne. Le label $Vert$ est centré sur la droite de l'espace, le label *Bleu* sur la gauche, le label *Rouge* sur la gauche plus haut que le label *Bleu*. Les données ont été créées pour que les labels *Bleu* et $Vert$ se chevauchent légèrement et que le label *Rouge* soit englobé complètement par le label *Bleu*. Les données des labels $Vert$ et *Bleu* ont une densité identique alors que le label *Rouge* est plus dense.

L'inégalité de Hoedffing se base sur une différence de deux sommes, l'une sont les données aléatoires, l'autre la somme des valeurs attendues, la ré-

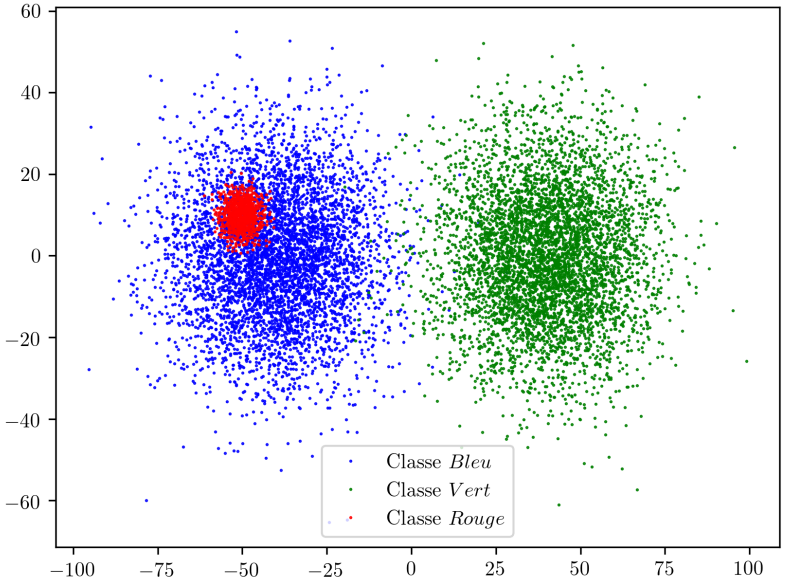


FIGURE 5.1 – Dataset synthétique *Rouge*, *Vert*, *Bleu*

flexion peut alors s'énoncer comme « Quelles sont les données dont on connaît une valeur approximative ou réelle et qui peuvent être exprimées sous forme de sommes et comparées entre elles ? ». La liste est assez courte. Dans le dataset modèle, les éléments connus sont les attributs et le label de chaque donnée. Il est également possible de calculer les probabilités qu'une donnée soit d'un label ou d'un autre, et ce en fonction de k. Il est également possible de calculer le score global du kNN en fonction de k.

Parmi ces données, certaines ne se peuvent pas être exprimées sous forme de sommes, et ne peuvent pas être utilisées pour avancer dans la réflexion. C'est le cas des labels et des attributs. Les labels car ils peuvent être non-numériques. Les attributs car ils sont, a priori, indépendants entre eux et les sommer ne fait aucun sens. Par contre, le score et les probabilités d'être un label ou un autre sont exprimables sous forme de sommes. En effet, le score est la moyenne des résultats bons ou mauvais (formellement 1 ou 0) de la classification d'un modèle de validation, et une moyenne étant une somme, le score remplit bien cette condition. Les probabilités sont également des moyennes, ce qui remplit également la condition. Par contre, le score s'avère

ne pas pouvoir être comparé à une autre somme ayant le même nombre d'éléments. Il n'y a donc pas moyen de l'utiliser avec l'inégalité de Hoeffding.

Les probabilités, quant à elles, remplissent toutes les conditions. Le tableau 4.1 montre que lorsque le kNN est appliqué à une donnée, le résultat est un ensemble de probabilités, et toutes ces probabilités ont été calculées avec un seul et même k. Donc, elles peuvent être utilisées les unes par rapport aux autres avec l'inégalité de Hoeffding. Une idée a alors germé dans les esprits : et si ces probabilités permettaient de déterminer un k à l'endroit où se trouve la donnée observée ? La réponse est oui ! En effet, en appliquant le kNN sur une donnée avec un k donné, il y a moyen de calculer la différence de probabilité des deux labels les plus probables. Cette différence correspond bien à un des termes de l'inégalité de Hoeffding.

Pour rappel, l'inégalité de Hoeffding(équation (2.3)) peut s'écrire comme ceci

$$|S_1 - S_2| \geq 2\sqrt{\frac{R^2 ln(\frac{1}{\delta})}{2n}} \tag{5.1}$$

$$S_l = \sum_{i=0}^{n} Y_{li} \tag{5.2}$$

où Y_{li} est le compte des données ayant le label l parmi les k plus proches voisins et divisé par k, S_1 la probabilité du label le plus probable, S_2 la probabilité du second label le plus probable (dans l'exemple de la tableau 4.1, $S_1 = 0.5$ et $S_2 = 0.25$), $R = 1$ car S est une probabilité dont le domaine est $[0, 1]$ et n égal au k utilisé pour calculer les probabilités. La comparaison peut alors se faire.

En faisant augmenter k, la borne de Hoeffding va diminuer, et à un moment donné l'inégalité donnera un résultat positif. A ce moment, le k a la valeur optimale pour déterminer combien de voisins il faut observer à cet endroit de l'espace multidimensionnel. La réflexion a donc été de généraliser ce calcul à tous les points du dataset modèle et ainsi déterminer un k optimum pour chacun. L'application du kNN doit être changée pour prendre en compte ces k locaux. Grâce à cette technique, le k du kNN peut maintenant être déterminé automatiquement et mathématiquement sans devoir faire les tests présentés dans le section 4.2.1 ce qui est, au final, l'objectif de l'algorithme.

Cette nouvelle manière de faire l'apprentissage, ainsi que l'application du kNN, est nommée : k local nearest neighbors (kLNN).

5.1 Définition de l'algorithme

L'algorithme du kLNN apporte deux modifications au kNN. La première se situe au niveau de l'apprentissage et la seconde lors de la classification.

5.1.1 Apprentissage du modèle

L'apprentissage du kNN consiste, comme expliqué dans la section 4.2.2, à déterminer un k et éventuellement optimiser la manière dont l'espace sera parcouru lors de la classification. L'apprentissage du kLNN modifie la détermination du k alors que l'optimisation du parcours de l'espace reste tout à fait valable et identique au kNN.

En pratique, l'apprentissage dans le kLNN ajoute un tableau de données de type entier de taille égale au nombre de données dans le dataset modèle. Ce tableau recevra la valeur du k local à utiliser lors de la classification.

L'algorithme 5.1 montre la nouvelle manière de déterminer les k. Cet algorithme doit être appliqué à toutes les données du dataset modèle. Dés lors, chaque donnée possède un k qui lui est propre. La procédure *Compute-MinHoeffdingN* calcule le nombre minimal d'éléments à observer avant que l'inégalité de Hoeffding ne soit utilisable (cf. section §2.3 et la figure 2.4). Cela est une optimisation du code qui permet de ne pas devoir appliquer le kNN pour les k trop petits. Pour information, la figure 5.2 montre l'évolution de cette valeur minimale en fonction du δ. La boucle d'évolution du k commence à la valeur minimale utile, mais termine à 10 fois cette valeur. Cette borne supérieure est totalement artificielle et permet de nouveau une optimisation. D'un point de vue pratique, il est rare qu'un k 10 fois supérieure au k minimum soit trouvé. Un exemple plus parlant est montré sur les figures 5.6 et 5.7. A la fin de l'algorithme, si aucun k n'a été validé, la procédure retourne $Undefined$. Cela signifie que la répartition des données est telle que l'inégalité de Hoeffding n'est jamais rencontrée. Cette information permet de déterminer certaines zones de l'espace où les labels sont tellement mélangées les uns aux autres qu'il n'est pas possible de classifier correctement une donnée en utilisant le kNN. La section 5.1.2 discute des manières possibles de prendre en compte ce cas.

Algorithme 5.1 Algorithme permettant de définir un k pour un élément du modèle donné avec une certaine probabilité $1 - \delta$

X_m	un modèle
X_c	un élément à classifier
$1 - \delta$	la probabilité de l'inégalité de Hoeffding
X_{mc}	$X_m \backslash \{X_c\}$

Procedure DefineK
  minK = *ComputeMinhoeffdingN*(δ)
  foreach k **in** [minK, minK∗10]
    probabilities = *GetProbabilities*(X_{mc}, X_c, k)
    probabilities = *Sort*(probabilities)
    diff =probabilities[0]−probabilities[1]
    hoeffding = *ComputeHoeffding*(1, δ, k)
    if diff > hoeffding
      return k
  return *Undefined*

ComputeMinHoeffdingN	Calcul n minimum en fonction de δ
Len	Calcul le nombre d'éléments dans la liste
GetProbabilities	Calcul les probabilités des différents labels de X_c dans X_{mc}avec k plus proches voisins
Sort	Trie la liste donnée
ComputeHoeffding	Calcul la borne de Hoeffding

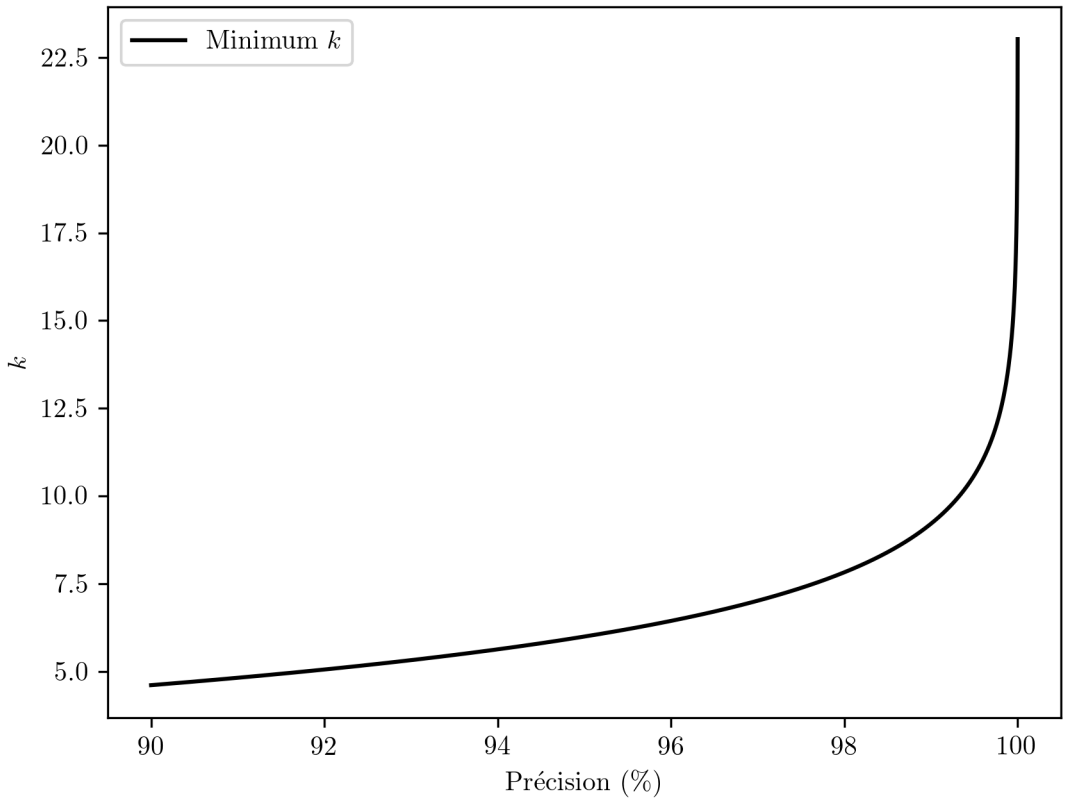


FIGURE 5.2 – k minimum en fonction de la précision de l'inégalité de Hoeffding

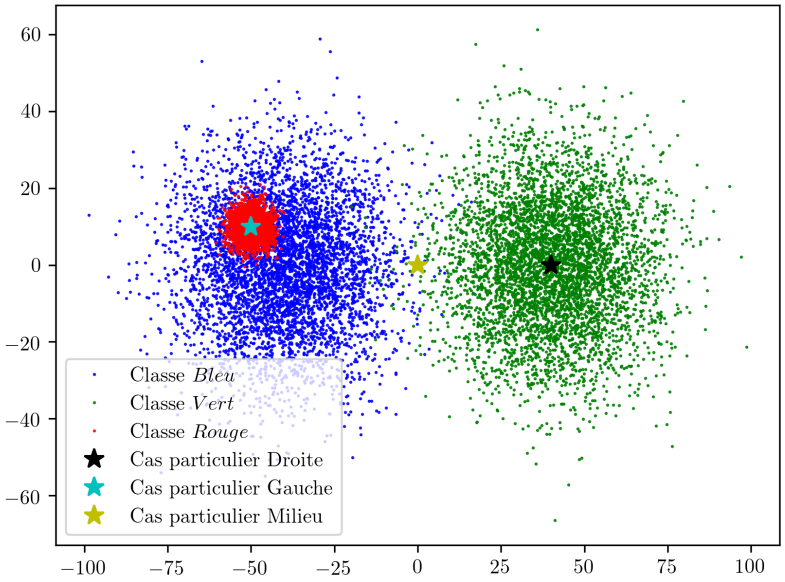


FIGURE 5.3 – Points particuliers sélectionnés pour analyser le comportement de l'algorithme d'apprentissage du kLNN

Exemple théorique

Pour observer le comportement de l'algorithme, avant même de mesurer ses performances, le dataset synthétique peut être utilisé (cf. figure 5.1). Dans l'espace de ce dataset, trois points particuliers sont observés. La figure 5.3 montre ces points, le premier, nommé *Droite*, est centré dans la zone du label $Vert$, le second, nommé *Gauche*, est centré dans la zone du label *Rouge* et le troisième, nommé *Milieu*, est situé à l'intersection des zones des labels$Vert$ et *Bleu*. Les mesures ont été faites avec comme probabilité dans l'inégalité de Hoeffding de 95%. Comme point de comparaison, le k déterminé par la méthode de la section 4.2.1 sur ce dataset est de 15.

Droite. Comme le montre la figure 5.4, l'inégalité de Hoeffding permet de déterminer $k = 6$. Cette valeur est la valeur minimale pour la probabilité utilisée (95%). Ce n'est pas surprenant, cette zone de l'espace est remplie du label $Vert$ et d'aucun autre, l'inégalité de Hoeffding se comporte alors comme expliqué à la section §2.3 et montré sur la figure 2.4, lorsque toutes

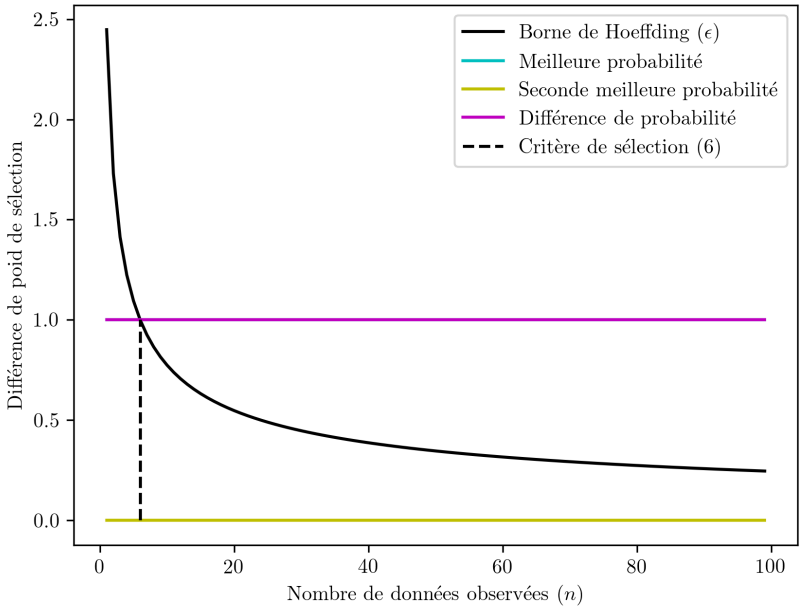


FIGURE 5.4 – Évolution de l'inégalité de Hoeffding pour le point particulier **Droite**

les observations sont identiques.

Gauche. Comme le montre la figure 5.5, l'inégalité de Hoeffding permet de déterminer $k = 10$. Cette valeur, supérieure à la valeur minimale montre que la zone est plus mélangée et plus difficile à interpréter qu'une zone ne comportant qu'un seul label, l'inégalité de Hoeffding se comporte alors comme expliqué à la section §2.3 et montré sur la figure 2.3. Néanmoins, le k est inférieur à la valeur du $k = 15$ calculé avec la méthode de la section 4.2.1, ce qui laisse à penser qu'un $k = 15$ est trop grand pour cette zone. Par contre, lorsque la probabilité augmente, par exemple à 99% alors, le k calculé devient 13, à 99.99%, le k passe à 24. En pratique, une probabilité de 95% est tout à fait raisonnable et est utilisé tout du long de ce mémoire.

Milieu. Comme le montre la figure 5.6 , l'inégalité de Hoeffding ne permet pas de déterminer le k, par convention k est définit comme *Undefined*. Cette non-détermination est dû au fait que deux labels se chevauchent, mais contrairement au point *Gauche*, les deux datasets ont la même densité, ce qui

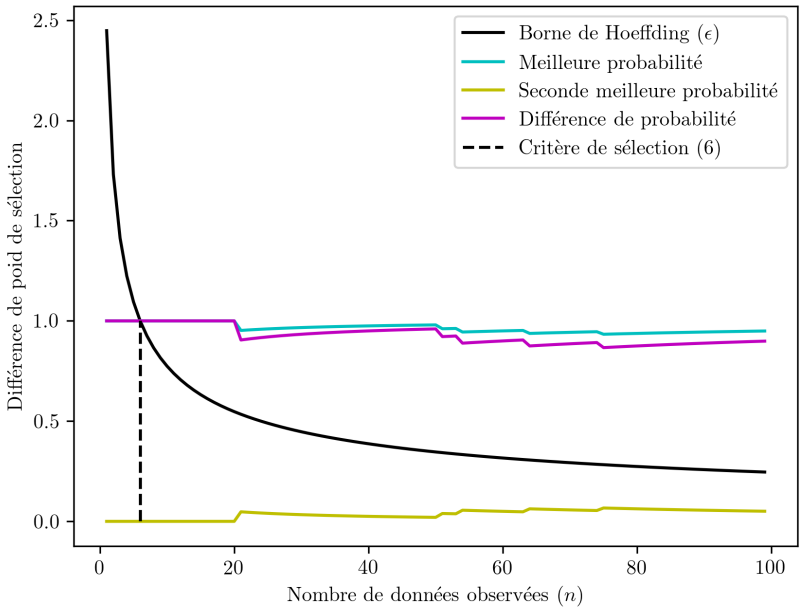


FIGURE 5.5 – Évolution de l'inégalité de Hoeffding pour le point particulier **Gauche**

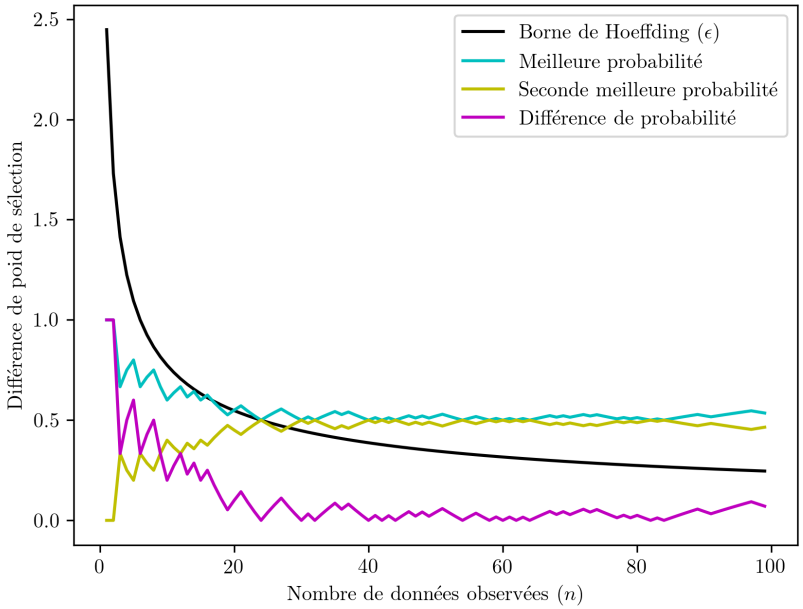


FIGURE 5.6 – Évolution de l'inégalité de Hoeffding pour le point particulier **Milieu**

fait que l'algorithme n'arrive pas à départager les deux labels. La figure 5.6 montre les 100 premières mesures, mais peut-être qu'en observant plus de données, il est possible de déterminer un k pour cette zone de l'espace. La figure 5.7 montre l'évolution des mesures avec comme seule limite la taille du dataset et l'algorithme n'arrive toujours pas à déterminer de valeur pour k. Dans la pratique, il n'est pas nécessaire de prendre en compte tout le dataset, le nombre de données maximale a prendre en compte a été fixé à $minK * 10$.

Observation globale. L'analyse des points particuliers n'apporte aucune réelle surprise et est bien conforme aux réflexions théoriques faites sur l'algorithme. L'étape suivante est de calculer le k sur tous les points du dataset. La figure 5.8 montre le dataset non plus avec une couleur par label, mais bien avec un dégradé de gris fonction des valeurs de k. Les k non définis sont dessinés en rouge. La probabilité utilisée dans l'inégalité de Hoeffding est ici de 99.99% afin de bien mettre en évidence les différentes zones. Sans grande surprise, les zones aux frontières entre deux labels ne possèdent pas de k valides. Et plus le label est isolé et plus le k est proche du k minimum.

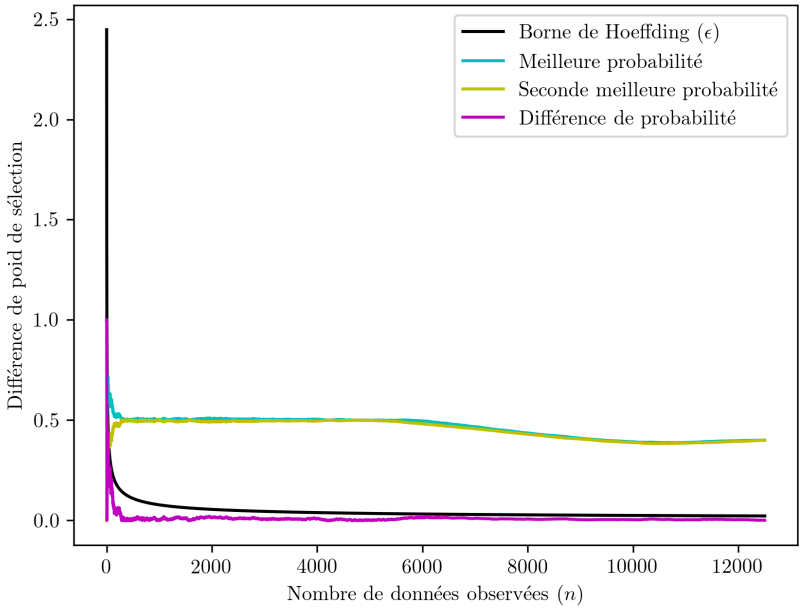


FIGURE 5.7 – Évolution de l'inégalité de Hoeffding pour le point particulier **Milieu** sur un grand nombre d'observations

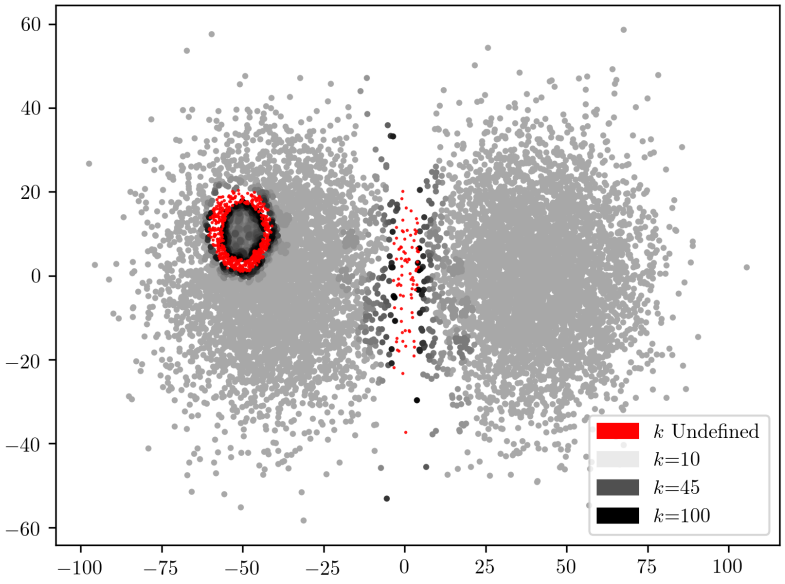


FIGURE 5.8 – Heatmap représentant l'apprentissage complet sur le dataset synthétique

Piste d'amélioration

Une première piste d'amélioration de cet algorithme est de ne pas rechercher les plus proches voisins à chaque fois. En effet, avec l'algorithme tel qu'il est décrit ici, pour un k donné, l'application du kNN va commencer par détecter les $k-1$ plus proches voisins puis seulement en ajouter un. Mais ce calcul a déjà été fait lors de la boucle précédente. Un algorithme itératif plus intelligent pourrait éviter cette duplication de calcul. Cette optimisation n'a pas été faite dans ce mémoire et est donc une amélioration possible.

Une seconde piste d'amélioration est de détecter les cas où la valeur minimale de k est sélectionnée et de ne pas utiliser cette valeur, mais d'utiliser $k = 1$. En effet, si la valeur minimale est sélectionnée, cela veut dire que toutes les observations précédentes donnent exactement le même résultat et le k minimal est donc trop élevé par rapport à ce qu'il pourrait être. Cela permettrait une amélioration lors de la classification en terme de temps d'exécution au minimum.

5.1.2 Application du modèle

La classification du kNN consiste, comme expliqué dans la section 4.2.1, à parcourir les points du dataset, calculer leurs distances au point à classifier, récupérer les k plus proches et compter les labels de ces k éléments pour en déterminer le label majoritaire.

La classification du kLNN ajoute une étape préliminaire à celle du kNN. Cette étape consiste à trouver le voisin le plus proche du point à classifier pour récupérer le k local lié à celui-ci. Ensuite, l'algorithme de classification du kNN est appliqué avec le k local trouvé.

k Indéfini

L'apprentissage permet, dans certains cas, de détecter des zones où le k ne peut pas être défini. Ces zones sont signalées par un $k = Undefined$. Dans le cas de la classification, cette valeur est remplacée par 10 fois le k minimum. Cette approche permet de comparer les résultats du kLNN à ceux du kNN sans devoir gérer de cas particuliers. Elle a le désavantage de fournir un résultat là où l'apprentissage signale que ce résultat n'est pas assuré, et risque d'apporter une certaine instabilité dans ces zones de l'espace.

Les observations sur le dataset synthétique montrent que le k est petit

Algorithme 5.2 Algorithme permettant de classifier un élément à partir d'un kLNN

X_m	un modèle
X_c	un élément à classifier
$1 - \delta$	la probabilité de Hoeffding

Procedure *Classifie*
  nearest= *GetNearest*(X_m, X_c)
  loclaK = *RetrieveK*(nearest)
  if localK == Undefined
    localK = ComputeMinHoeffdingN(δ)
  return kNN.Classifie(X_m, X_c, localK)

GetNearest	retourne le plus proche voisin de $X_c$dans X_m
RetrieveK	retourne le k lié à nearest
k**NN.Classifie**	retourne le label de $X_c$dans X_m en utilisant les kNN avec $k = localK$

dans les zones où il n'y a qu'un label proche et grandit lorsque deux (ou plus) labels se chevauchent. Vu la formule de l'inégalité de Hoeffding (équation (2.3)), lorsque ϵ tend vers 0, n tend vers l'infini (voir figure 5.9). Le choix de 10 fois le k minimum reflète cette tendance à l'infini et la borne au k maximum traité dans l'algorithme d'apprentissage.

Pistes d'amélioration

Dans la discussion sur les pistes d'amélioration de l'apprentissage, une possibilité d'amélioration est d'assigner 1 à k lorsque la borne de Hoeffding donne comme valeur de k le k minimum possible. Alors, l'algorithme de classification pourrait se simplifier et renvoyer directement le label lié à l'élément *nearest* du dataset sans devoir le re-parcourir entièrement. Cette modification devrait améliorer le temps d'exécution du kLNN sans pour autant diminuer sa précision.

La valeur *Undefined* pour k pourrait être gérée autrement :

— En déterminant une valeur de k permettant d'atteindre une précision maximum. Cette manière de faire demande une nouvelle réflexion sur la manière de déterminer ce k. Une piste serait de chercher une valeur de k de la même manière que pour le kNN mais dans une zone restreinte de l'espace. Sans cette modification, les résultats obtenus sur les datasets réels (cf. chapitre 6) sont assez bons, et donc elle n'a pas

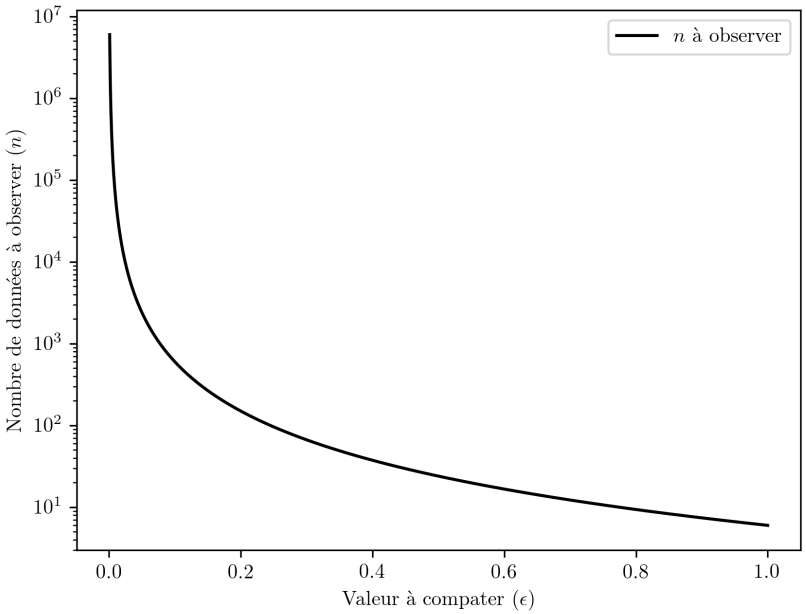


FIGURE 5.9 – Nombres d'observation nécessaires pour valider l'inégalité de Hoeffding en fonction de l'ϵ observé

été abordée lors de ce mémoire.

— En interdisant (via une exception par exemple) de faire une classification sur une zone de l'espace signalée comme *Undefined*. Cette manière de faire n'a pas été retenue dans le cadre de ce mémoire car elle casse la compatibilité entre le kNN et le kLNN. En effet, actuellement, l'implémentation réalisée du kLNN peut être utilisée de manière totalement transparente en lieu et place du kNN.

— En utilisant le nombre de k ayant cette valeur comme indicateur d'adéquation entre le dataset traité et l'algorithme des plus proches voisins. De cette manière, il serait possible de définir si les plus proches voisins est le bon algorithme pour classifier un dataset donné.

Chapitre 6

Le kLNN : expériences pratiques

La description théorique du kLNN présentée dans le chapitre 5 montre des résultats théoriques intéressants. Ce chapitre se penche sur des mesures empiriques réalisées lors de ce mémoire pour vérifier si la théorie rencontre bien la pratique.

6.1 Description du setup expérimental

6.1.1 La logique utilisée

L'algorithme figure 6.1 explicite la procédure utilisée pour générer les mesures présentées dans ce chapitre. Il utilise la technique de *k-Fold cross validation*[14]. Ici, le nombre de sous-divisions est fixé à 10.

Une fois cette procédure réalisée sur tous les datasets, un ensemble de données est disponible pour analyse. Les données brutes, c'est-à-dire l'ensemble des scores obtenus par la procédure, sont trop nombreux pour les présenter directement dans ce document. Par contre, des statistiques sur celles-ci seront exposées et analysées dans la suite de ce chapitre.

6.1.2 Les datasets utilisés

Pour couvrir un ensemble de cas réels important, plusieurs datasets ont été étudiés. Ceux-ci on été récupéré via le site « UCI Machine Learning Repository »[5]. Le tableau suivant reprend le détail des datasets étudiés :

Algorithme 6.1 Algorithme de validation *k-Fold* utilisé pour la validation des algorithmes.

X_m	un modèle
n	le nombre de sous-divisions à faire
kNN	l'algorithme à valider

Procedure *Validate*
  X_{mn}= *SplitDataset*(X_m, n)
  scores = []
  foreach X_{mi} **in** X_{mn}
    $X_{mtrain} = X_m - X_{mi}$
    Train(kNN, X_{mtrain})
    score = *Score*(kNN, X_{mi})
    Add(scores, score)
  return scores

SplitDataset	divise le dataset $X_m$en k sous-parties de taille identique
Train	entraine l'algorithme avec le sous-dataset X_{mtrain}
Score	calcul la précision du kNN sur X_{mi}
Add	ajoute une donnée au tableau donné

Dénomination	Abréviation	Instances	Attributs	Labels
Synthétique(5)	Synth.	11000	2	3
Ionosphere[20]	Iono	351	34	2
Wall Following 2[8]	Wall2	5456	2	4
Wall Following 4[8]	Wall4	5456	4	4
Wall Following 24[8]	Wall24	5456	24	4
Red Wine Quality[3]	RedW	1599	11	10
White Wine Quality[3]	WhiteW	4898	11	10
Breast Cancer Wisconsin	Bcw	699	9	2
Wdbc	Wdbc	569	30	2

6.2 Résultats

Le kLNN permet la détermination automatique des valeurs des k à utiliser lors de la classification d'une donnée. Cela permet d'éviter de devoir déterminer le k une fois l'apprentissage réalisé. Les résultats présentés ici montrent l'impact qu'une telle détermination automatique a sur la précision de l'algorithme. Ils montrent également les valeurs que peut prendre le k.

Les mesures de précision réalisées sont données dans le tableau 6.1. Ce

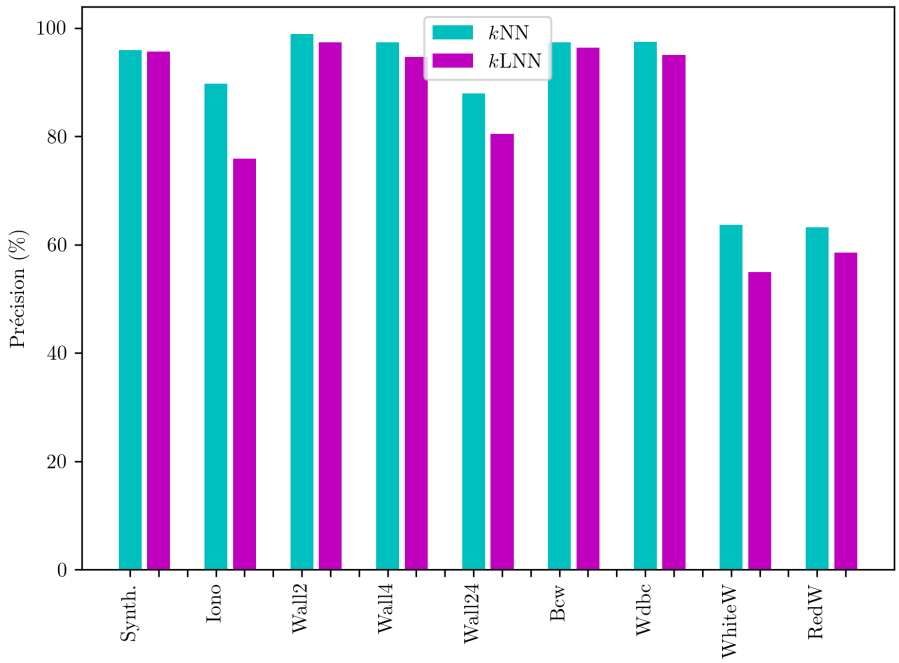


FIGURE 6.1 – Mesures expérimentales de la précision : kNN vs kLNN

tableau montre la moyenne, le minimum, le maximum et l'écart-type de la précision pour le kNN et le kLNN. La figure 6.1 montre les valeurs moyennes du tableau sous forme d'un graphique. Les mesures des valeurs de k réalisées sont données dans le tableau 6.2. Ce tableau montre la moyenne de ces valeurs pour le kNN et le kLNN.

Lorsque l'on parle de précision, le kLNN se comporte assez bien, ses résultats en terme de précisions sont proches des précisions du kNN. Trois datasets ont tout de même une chute de précision proche de 10% : ionosphere, white wine quality et wall followong 24. Une explication possible est du à l'augmentation de la valeur du k, en effet, ces trois datasets sont ceux où le k augmente le plus significativement. Les améliorations proposées à la fin du chapitre 5 pourraient apporter des réponses à cette chute, principalement la gestion des k non défini.

Lorsqu'il s'agit du k, il y a de gros changements. Ces changements s'expliquent avec l'utilisation de l'inégalité de Hoeffding qui impose un nombre minimum d'observations, et donc un k minimum supérieure à 1, pour pouvoir assurer, à une probabilité près, que le choix est le bon. Néanmoins, il est

Dataset		Précision kNN	Précision kLNN
Synth.	Moyenne	95.91	95.65
	Minimum	95.22	64.63
	Maximum	96.68	96.50
	Écart-type	0.003	0.004
Iono	Moyenne	89.76	75.85
	Minimum	84.28	59.15
	Maximum	94.36	90.00
	Écart-type	0.026	0.055
Wall2	Moyenne	98.93	97.34
	Minimum	97.89	95.96
	Maximum	99.54	98.26
	Écart-type	0.002	0.004
Wall4	Moyenne	97.35	94.65
	Minimum	96.42	92.66
	Maximum	98.25	96.51
	Écart-type	0.004	0.008
Wall24	Moyenne	87.97	80.46
	Minimum	85.43	77.20
	Maximum	90.65	82.40
	Écart-type	0.009	0.012
RedW	Moyenne	63.25	58.58
	Minimum	57.81	52.5
	Maximum	66.25	63.43
	Écart-type	0.018	0.024
WhiteW	Moyenne	63.67	54.98
	Minimum	60.98	51.63
	Maximum	67.04	58.26
	Écart-type	0.013	0.014
Bcw	Moyenne	97.38	96.38
	Minimum	94.96	91.42
	Maximum	99.28	100
	Écart-type	0.011	0.015
Wdbc	Moyenne	97.49	95.01
	Minimum	92.92	90.35
	Maximum	100	98.24
	Écart-type	0.016	0.02

TABLE 6.1 – Mesures expérimentales de la précision : kNN vs kLNN

Dataset	k moyen kNN	k moyen kLNN
Synth.	32.7	7.3
Iono	2.4	8.8
Wall2	1.6	6.5
Wall4	1.2	7.4
Wall24	1.2	9.4
RedW	3.6	10.1
WhiteW	1	10
Bcw	7.5	7
Wdbc	4.5	7.5

TABLE 6.2 – Mesures expérimentales du nombre de voisins à prendre en compte (k) : kNN vs kLNN

possible d'isolé plusieurs cas de changement :

k du kNN élevé. Dans ce cas, la moyenne des k du kLNN peut réduire cette valeur. Cela s'explique par la taille des datasets. En effet plus un dataset est grand et plus il y a de chances de trouver des zones ne comportant qu'un label et donc générant des valeurs de k minimum. La moyenne est donc influencée par ces zones de l'espace. C'est ce qu'il se passe pour le dataset synthétique et celui sur le vin rouge. L'objectif du kLNN est bien atteint ici. Spécifier un k local à chaque élément du dataset permet de prendre en compte la répartition des labels et de ne pas perdre de temps à trouver un grand nombre de voisins là où ce n'est pas nécessaire.

k du kNN très bas voir égal à 1. Dans ce cas, par contre, le kLNN ne se comporte pas bien. Cela s'explique assez facilement avec le k minimum lié à l'inégalité de Hoeffding. En effet, si la majorité des points du dataset est bien isolée d'un point de vue des labels, alors ces points auront un k égal au k minimum. Les points aux frontières auront un k supérieur. Et donc, la moyenne sera toujours légèrement supérieure au k minimum. C'est ce qui est observé ici.

k du kNN moyen. Dans le cas où le k calculé pour le kNN est proche du k minimum, alors le kLNN calcule des k locaux proche également du k minimum. Et donc, les valeurs moyenne sont proches également.

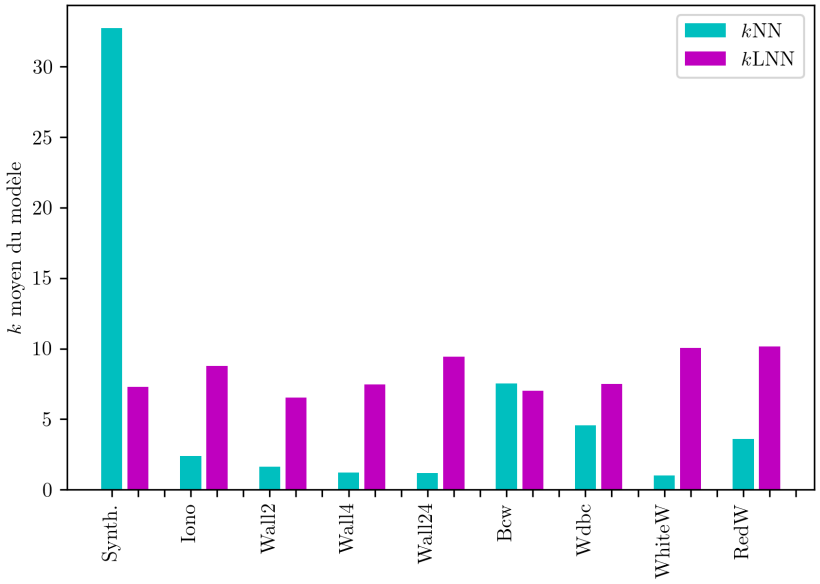


FIGURE 6.2 – Mesures expérimentales du nombre de voisins à prendre en compte (k) : kNN vs kLNN

Un phénomène remarquable et non prévu lors des réflexions théoriques est la relative constance des k calculé par le kLNN. Malgré le fait que les datasets utilisés s'appliquent à des domaines très différents les uns des autres, que la quantité de données contenus dans ceux-ci ainsi que le nombre d'attributs par donnée sont très différents d'un dataset à l'autre, toutes les moyennes sont comprises entre le k minimum (6 dans le cas des tests ici) et 10.

Remarque

Dans la littérature, l'article [1] propose également une technique pour déterminer des k locaux. Il expose des résultats intéressants. En effet, on peut y voir que sur le dataset Ionosphere, il améliore la précision de plus de 1%. De plus, les k minima calculé est de 1 ou 2 alors que ceux du kLNN est de 6. Et les k maxima ne montent que jusqu'à 24, dans l'implémentation du kLNN, le k maximum est limité à 60 pour une probabilité de 95% comme utilisé lors des expériences. L'article en question définit toujours un k et ne propose pas de k non définit comme le fait le kLNN.

6.3 Conclusion

Le plus grand avantage du kLNN par rapport au kNN est de permettre de définir le k sans devoir passer par une phase de validation/cross validation/... pour déterminer le k. De plus, le kLNN se comporte assez bien d'un point de vue précision par rapport au kNN. Il ne l'améliore pas, mais génère des valeurs moyennes de k assez constantes et pas très éloignées du k minimum imposé par la borne de Hoeffding (entre 6 et 10). Ceci lui donne un intérêt substantiel sur des datasets où le k utilisé dans le kNN est élevé, car en moyenne la classification pourra se contenter d'un plus petit nombre de voisins à prendre en compte et donc ira plus vite.

Chapitre 7

Le kNN réduit

Lors des recherches et des expérimentations sur le kLNN, il a été remarqué que dans certaines zones du dataset, le k calculé est le même pour plusieurs points. Pourquoi ne pas tenter de fusionner ces points pour ainsi réduire le nombre d'éléments dans le dataset appris ? Cette question est la base de la suite de ce mémoire. De nouveau, la réflexion a été faite en tentant d'utiliser la borne de Hoeffding.

Après réflexion, le principe de fusion des points est reformulé comme étant un principe d'ajout conditionnel de ceux-ci. L'idée est de prendre les points du dataset modèle un à un et de déterminer si le fait de l'ajouter au modèle apporte de l'information ou pas. Si l'ajout n'apporte pas d'information utile, alors il est n'est pas ajouté. Pour ce faire, comme pour le kLNN, il faut trouver les deux sommes qui seront utilisées dans la différence du terme de gauche de l'inégalité de Hoeffding (cf. section §2.3). La piste suivie découle d'une interprétation de la technique d'apprentissage du kLNN, lors de cet apprentissage, l'algorithme cherche et fixe un k pour chaque élément du dataset. En imaginant qu'un kLNN est appris, ses éléments possèdent donc un k à l'exception des zones *Undefined*. En ajoutant un élément dans le dataset, s'il est proche d'autres éléments possédant déjà un k, cela veut dire que, dans cette zone de l'espace, les données sont suffisantes (à une probabilité près) pour prendre la bonne décision concernant le label correspondant à cette zone. Et donc, ce nouvel élément est inutile puisque la bonne décision sera déjà prise sans utiliser l'information qu'il apporte. Donc, c'est un élément qui n'apporte pas d'information utile, il ne doit pas être ajouté. Si maintenant, les autres éléments proches ont leur k à *Undefined* cela veut dire qu'il n'y a pas assez d'information dans cette zone de l'espace pour prendre une

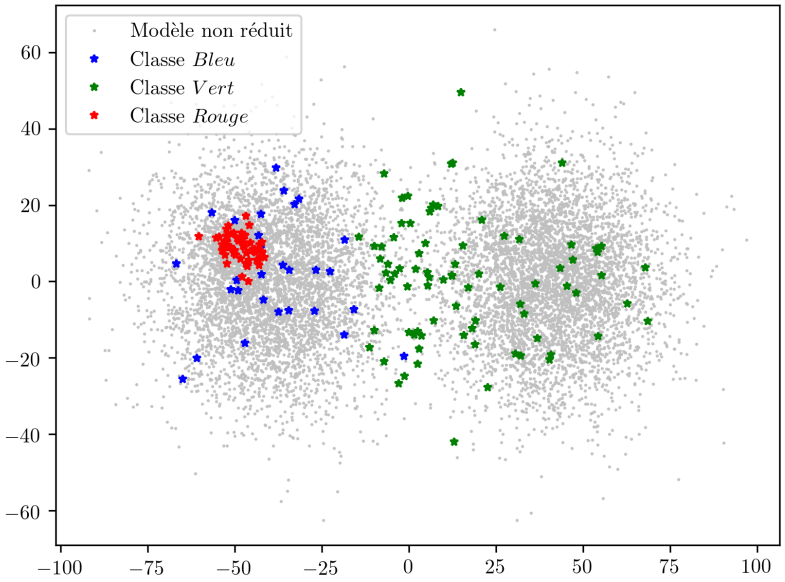


FIGURE 7.1 – Exemple d'application du kNN Réduit sur le dataset synthétique

décision. Et donc, il faut ajouter le nouvel élément.

L'algorithme proposé dans ce chapitre est un dérivé de celui présenté juste ci-dessus. A la place de regarder si les éléments proches ont un k défini ou pas, l'algorithme tente de déterminer un k pour l'élément. Il le fait à la manière du kLNN. S'il y arrive, cela veut dire qu'il y a assez d'information dans la zone et que l'élément ne doit pas y être ajouté. S'il n'y arrive pas, alors il manque de l'information, et l'élément est ajouté. Une fois cet algorithme appliqué, le dataset modèle est réduit et ne contient plus que les points apportant de l'information utile au système. La figure 7.1 montre le dataset d'exemple réduit avec le dataset original en gris clair pour comparaison.

A partir du dataset ainsi réduit, deux possibilités sont envisagées :

— Le dataset ainsi réduit est utilisé dans le cadre d'un kNN, il faut alors déterminer un k global et vérifier la précision obtenue.

— Le dataset ainsi réduit est utilisé avec l'algorithme du kLNN. Il faut alors repasser sur tous les points pour déterminer les k locaux. Cette étape est nécessaire, car comme expliqué, les points gardés dans le dataset n'ont pas de k défini.

Algorithme 7.1 Algorithme permettant de définir si un élément d'un dataset modèle doit être pris en compte ou pas lors de l'apprentissage du kNN Réduit

X_m	un dataset
X_l	le modèle en cours d'apprentissage
X_c	une donné tel quel $X_c \in X_m$et $X_c \notin X_l$
$1 - \delta$	la probabilité de Hoeffding

Procedure *HaveToAdd*
  k= *DefineK*(X_l, X_c,δ)
  if k = Undefined
    return Yes
  else
    return No

DefineK définit la valeur de k (cf. algorithme 5.1

Les deux cas sont étudiés dans le chapitre 8. L'application de la classification une fois l'apprentissage réalisé est exactement le même que pour le kNN ou le kLNN selon la dernière phase de l'apprentissage choisi.

7.1 Définition de l'algorithme

L'apprentissage du kNN Réduit consiste à prendre les éléments du dataset modèle un à un et de tenter de leur définir un k. S'il y arrive, l'élément est ignoré, sinon il est ajouté au dataset appris.

Pour tenter de définir le k, l'algorithme 7.1 est appliqué. Il montre la logique utilisée pour déterminer si un élément est à garder ou pas. La procédure *DefineK* est celle décrite dans la partie d'apprentissage du kLNN (5.1)

Une fois cet algorithme appliqué, comme dit dans l'introduction de ce chapitre, il faut soit trouver un k global, soit des k locaux. La manière de faire pour le premier cas est décrite dans la section §4.2, pour le second dans le chapitre 5.

7.1.1 Piste d'évolution

L'idée de vérifier si une donnée doit être ajoutée ou pas si les voisins possèdent un k défini ou pas serait intéressante à implémenter malgré les diverses questions qu'elle soulève. L'avantage serait de ne pas avoir à faire la

seconde passe de détermination des k, puisqu'ils seraient déjà déterminé par l'algorithme.

Chapitre 8

Le kNN Réduit : expériences pratiques

La description théorique et l'exemple sur le dataset synthétique du kNN Réduit présenté dans le chapitre 7 montre des résultats encourageants. Ce chapitre se penche sur des mesures empiriques réalisées lors de ce mémoire pour vérifier si les premiers résultats théoriques sont bien confirmés par la pratique.

8.1 Description du setup expérimental

La description du setup expérimental est pratiquement la même que pour les expériences pratiques sur le kLNN de la section §6.1. La procédure est la même pour vérifier la précision du kNN Réduit, une mesure supplémentaire est faite sur la taille du dataset appris. En effet, le but de cet algorithme est de réduire la taille de ce dataset.

Les datasets utilisés pour réaliser les expérimentations sont les mêmes que ceux pour le kLNN.

8.2 Résultats

Le kNN Réduit permet de réduire la quantité de données qui seront utilisées lors de la classification d'une donnée.

Les mesures de précision réalisées sont données dans le tableau 8.1. Ce tableau montre les moyennes, minima, maxima et écart-type pour le kNN, le

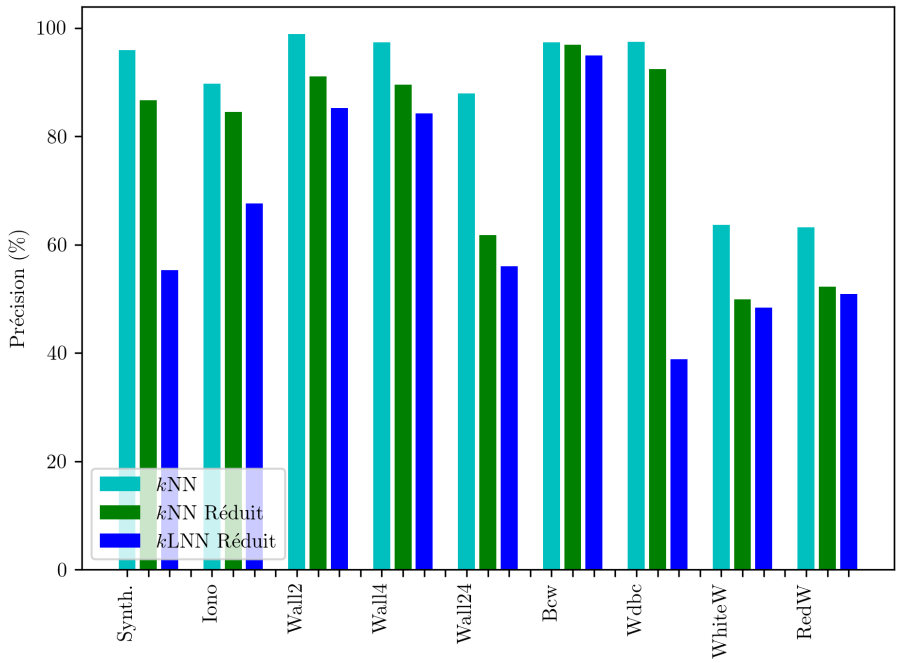


FIGURE 8.1 – Mesures expérimentales de la précision pour kNN, kNN Réduit et kLNN Réduit

kNN Réduit et le kNN Réduit combiné avec le$k$LNN. La figure 8.1 montre les valeurs moyennes du tableau sous forme graphique. Les mesures de tailles réalisées sont données dans le tableau 8.2. Ce tableau montre les moyennes, minima et maxima pour le kNN Réduit. Il donne la valeur originale pour comparaison. La figure 8.2 montre les valeurs moyennes du tableau sous forme de graphique.

Les résultats sur la précision ne sont pas extraordinaires. En effet, la précision du kNN Réduit par rapport au kNN systématiquement inférieure. Ceci n'est pas surprenant étant donné la réduction significative de la taille des dataset. L'objectif de réduction est bien atteint. La comparaison entre le kNN Réduit et le kLNN montre des résultats encore moins bons d'un point de vue précision. Cela s'explique par le fait que le k minimum imposé par l'inégalité de Hoeffding dans le kLNN peut avoir un effet désastreux sur un dataset très petit. Cette observation est intéressante et permet de dire qu'en l'état, il ne fait pas utiliser le kLNN sur des petits datasets. Il a d'ailleurs été remarqué à la section §6.2 qu'ionosphere ne se comportait déjà pas très bien, or c'est le dataset le plus petit testé.

Dataset	Précisions	kNN	kNN Réduit	kLNN Réduit
Synth.	Moyenne	95.91	86.70	55.26
	Minimum	95.22	81.27	43.77
	Maximum	96.68	93.36	82.77
	Écart-type	0.003	0.031	0.131
Iono	Moyenne	89.76	84.50	67.64
	Minimum	84.28	74.28	31.42
	Maximum	94.36	94.28	85.71
	Écart-type	0.026	0.043	0.111
Wall2	Moyenne	98.93	91.05	85.23
	Minimum	97.89	83.86	62.14
	Maximum	99.54	97.15	95.87
	Écart-type	0.002	0.038	0.066
Wall4	Moyenne	97.35	89.51	84.23
	Minimum	96.42	86.43	75.16
	Maximum	98.25	92.94	90.10
	Écart-type	0.004	0.015	0.032
Wall24	Moyenne	87.97	61.74	56.04
	Minimum	85.43	41.43	14.20
	Maximum	90.65	81.85	76.99
	Écart-type	0.009	0.136	0.162
RedW	Moyenne	63.25	52.28	50.89
	Minimum	57.81	39.06	39.06
	Maximum	66.25	65.00	62.81
	Écart-type	0.018	0.074	0.075
WhiteW	Moyenne	63.67	49.90	48.36
	Minimum	60.98	43.67	41.67
	Maximum	67.04	55.81	54.59
	Écart-type	0.013	0.038	0.037
Bcw	Moyenne	97.38	96.87	94.95
	Minimum	94.96	72.85	60.00
	Maximum	99.28	99.28	98.57
	Écart-type	0.011	0.32	0.047
Wdbc	Moyenne	97.49	92.39	38.81
	Minimum	92.92	77.19	26.31
	Maximum	100	96.46	89.47
	Écart-type	0.016	0.028	0.101

TABLE 8.1 – Mesures expérimentales de la précision : kNN vs kNN Réduit et kLNN Réduit

Dataset	Taille	Taille d'apprentissage kNN	kNN Réduit
Synth.	Moyenne	8800	40
	Minimum		29
	Maximum		117
Iono	Moyenne	280	62.3
	Minimum		38
	Maximum		100
Wall2	Moyenne	4364	323.5
	Minimum		81
	Maximum		386
Wall4	Moyenne	4364	594
	Minimum		508
	Maximum		666
Wall24	Moyenne	4364	671.3
	Minimum		42
	Maximum		1794
RedW	Moyenne	1279	300.4
	Minimum		20
	Maximum		696
WhiteW	Moyenne	3918	1722
	Minimum		32
	Maximum		904
Bcw	Moyenne	559	57.8
	Minimum		11
	Maximum		77
Wdbc	Moyenne	455	16.2
	Minimum		11
	Maximum		22

TABLE 8.2 – Mesures expérimentales de la taille du dataset résultat pour le kNN Réduit

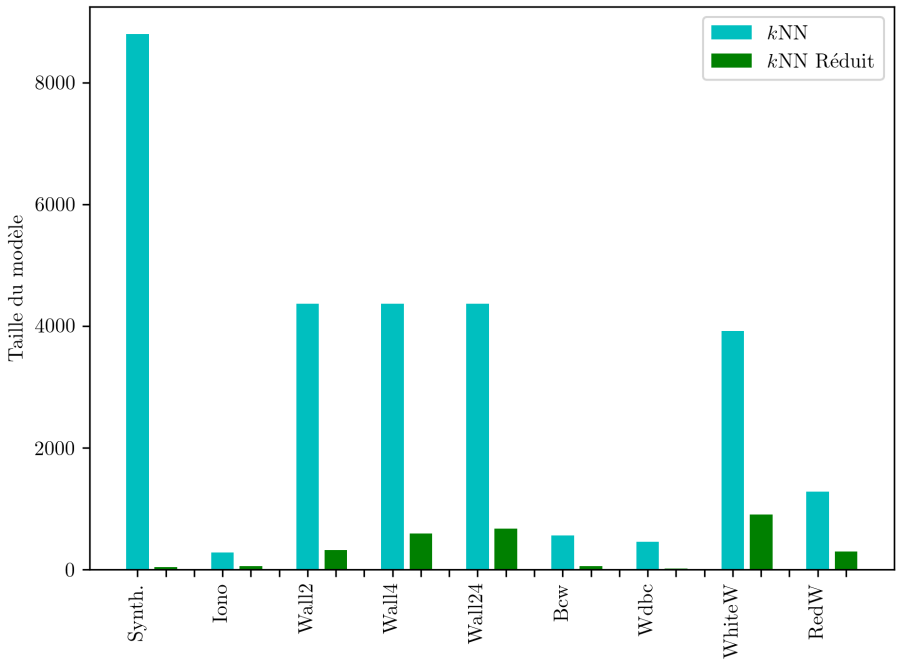


FIGURE 8.2 – Mesures expérimentales de la taille du dataset appris :kNN vs kNN Réduit

Les résultats sur la taille des datasets est, par contre, lui très bon. En effet, le facteur de réduction de taille des dataset est de 3 à 28 (en dehors du dataset synthétique qui se prête un peu trop bien à l'exercice avec son facteur de 220). Au final, la combinaison précision et taille du dataset est plutôt bonne, sachant qu'il est sûrement possible d'améliorer la précision en améliorant le kLNN aux petits datasets.

8.3 Conclusions

L'algorithme présenté ici réduit de manière impressionnante la taille des données à utiliser lors de la classification. Le coût est une réduction de la précision obtenue. Néanmoins, les pertes de précisions observées sont relativement limitées par rapport à la diminution de la taille des données. L'algorithme du kNN Réduit remplit donc bien son rôle.

La combinaison du kNN Réduit et du kLNN met en évidence une limitation du kLNN qui est de ne pas bien se comporter sur des datasets très petits.

De nouveau, les pistes d'améliorations proposées sur celui-ci pourraient grandement améliorer la précision de la combinaison.

Chapitre 9

Conclusion

Ce mémoire s'est penché sur l'inégalité de Hoeffding telle qu'utilisée par Pedro Domingos et Goeff Hulten[6] dans le cadre des arbres de décision. Il explique comment l'inégalité de Hoeffding[10] permet de résoudre certains défis posés par le big data. Les défis relevés sont liés au volume des données en ce sens où l'utilisation de l'inégalité de Hoeffding permet de ne pas devoir mémoriser toutes les données apprises, mais seulement un compte de celles-ci. De plus, l'inégalité de Hoeffding permet de réduire la taille des arbres de décision. Ceci a deux effets principaux, le premier étant d'éviter l'overfitting des arbres créés, le second étant d'augmenter l'efficacité en terme de temps de calcul lors de la classification d'une donnée. Une variante de cette utilisation de l'inégalité de Hoeffding est présentée pour les arbres de décision afin de prendre en compte le concept de drift du domaine d'application des données. Il survole également l'utilisation de l'inégalité de Hoeffding dans d'autres algorithmes proposés par Perdo Domingos et Goeff Hulten[12, 19, 7, 11]. Ce mémoire met également en évidence les travaux de Leszek Rutkowski, Lena Pietruczuk, Piotr Duda et Maciej Jaworski[18] remettant en cause la bonne utilisation de l'inégalité de Hoeffding dans les arbres de décision selon Pedro Domingos et Goeff Hulten, et proposant l'utilisation de l'inégalité de McDiarmid[15] comme alternative.

Ce mémoire propose ensuite deux nouvelles applications de l'inégalité de Hoeffding dans le cadre de l'algorithme de classification des plus proches voisins.

L'objectif de la première proposition est de déterminer de manière automatique et mathématique le nombre de voisins à prendre en compte lors de la classification d'une donnée. Pour ce faire, il calcule, grâce à l'inégalité de

Hoeffding, un nombre de voisins à prendre en compte pour chaque élément du modèle. Ensuite, ce nombre est utilisé au moment de la classification. Ce nouvel algorithme, baptisé kLNN pour *k-local nearest neighbors*, détermine des k différents en fonction de la zone de l'espace du modèle. Ainsi une zone du modèle où il n'y a qu'un label aura un k faible, alors que les zones contenant plusieurs labels auront des k plus grand. Il met également en évidence, grâce aux k non définissables, des zones du modèle où il n'y a pas moyen de déterminer avec certitude la classe d'une donnée présente à cet endroit. La gestion de ces zones pourrait faire l'objet d'études futures intéressantes. Une autre future réflexion serait de se pencher sur la gestion des zones où le k est minimal par rapport à l'inégalité de Hoeffding, et voir s'il n'y aurait pas moyen de réduire encore le k. La seconde application proposée par ce mémoire met en évidence que le kLNN ne se comporte pas très bien sur de petits datasets, ce qui est explicable par le fait que l'inégalité de Hoeffding ne permet de prendre de décision qu'après avoir fait un certain nombre d'observations. Si le dataset est petit, ces observations peuvent rapidement amener un effet négatif sur la précision du modèle appris.

La seconde application proposée a comme objectif de réduire la quantité de données à mémoriser dans le modèle tout en limitant au maximum la perte de précision du kNN. Pour ce faire, l'inégalité de Hoeffding est utilisée pour déterminer si l'ajout d'une donnée dans le modèle apporte une information utile. Si ce n'est pas le cas, l'information n'est pas ajoutée. Cet algorithme montre des résultats prometteurs avec des facteurs de réduction allant de 3 à 28, et ce avec des pertes de précision limitées à 1% au mieux et à 13% au pire.

La performance des deux applications ci-dessus a été vérifée informatiquement sur des datasets de référence, ce qui a confirmé leur efficacité. À notre connaissance, ces applications de l'inégalité de Hoeffding constituent une contribution originale et effectivement utile à la résolution des problèmes de classification par la méthode des plus proches voisins.

Bibliographie

[1] Oren Anava and Kfir Levy. k-nearest neighbors : From global to local. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4916–4924. Curran Associates, Inc., 2016.

[2] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9) :509–517, September 1975.

[3] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4) :547 – 553, 2009. Smart Business Networks : Concepts and Empirical Evidence.

[4] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1) :1–38, 1977.

[5] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017.

[6] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 71–80, New York, NY, USA, 2000. ACM.

[7] Pedro Domingos and Geoff Hulten. Learning from infinite data in finite time. In *Advances in neural information processing systems*, pages 673–680, 2002.

[8] A. L. Freire, G. A. Barreto, M. Veloso, and A. T. Varela. Short-term memory mechanisms in neural network learning of robot navigation tasks : A case study. In *2009 6th Latin American Robotics Symposium (LARS 2009)*, pages 1–6, Oct 2009.

[9] Jerome H Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3) :209–226, 1977.

[10] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301) :13–30, 1963.

[11] Geoff Hulten and Pedro Domingos. Vfml–a toolkit for mining high-speed time-changing data streams. *Software toolkit*, page 51, 2003.

[12] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 97–106, New York, NY, USA, 2001. ACM.

[13] Ashraf M. Kibriya and Eibe Frank. An empirical comparison of exact nearest neighbour algorithms. In Joost N. Kok, Jacek Koronacki, Ramon Lopez de Mantaras, Stan Matwin, Dunja Mladenič, and Andrzej Skowron, editors, *Knowledge Discovery in Databases : PKDD 2007*, pages 140–151, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[14] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada, 1995.

[15] Colin McDiarmid. On the method of bounded differences. *Surveys in combinatorics*, 141(1) :148–188, 1989.

[16] Stephen M Omohundro. *Five balltree construction algorithms*. International Computer Science Institute Berkeley, 1989.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn : Machine learning in Python. *Journal of Machine Learning Research*, 12 :2825–2830, 2011.

[18] L. Rutkowski, L. Pietruczuk, P. Duda, and M. Jaworski. Decision trees for mining data streams based on the mcdiarmid's bound. *IEEE Transactions on Knowledge and Data Engineering*, 25(6) :1272–1279, June 2013.

[19] Rahul Shah, Shonali Krishnaswamy, and Mohamed Medhat Gaber. Resource-aware very fast k-means for ubiquitous data stream mining.

In *In Proceedings of 2nd International Workshop on Knowledge Discovery in Data Streams, to be held in conjunction with the 16th European Conference on Machine Learning (ECML'05) and the 9th European Conference on the Principals and Practice of Knowledge Disc*, 2005.

[20] V G Sigillito, S P Wing, L V Hutton, and K B Baker and. Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Tech. Dig*, vol. 10 :262–266, 1989. in.